1972

An AN/UYK-7 interpreter implemented on the IBM System/360

Powell, Frederick Clair; Webb, Allen Borden; King, James Richardson.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/15989

AN AN/UYK-7 INTERPRETER IMPLEMENTED ON THE IBM SYSTEM/360

Frederick Clair Powell

NAVAL POSTGRADUATE SCHOOL

Monterey, California

AN AN/UYK-7 INTERPRETER IMPLEMENTED
ON THE IBM SYSTEM/360

by

Frederick Clair Powell
Allen Borden Webb
James Richardson King

Thesis Advisor: Raymond H. Brubaker, Jr.

December 1972

Approved for public release; distribution unlimited

An An-UYK/7 Interpreter
Implemented on
The IBM System/360

by

Frederick Clair Powell
Lieutenant Commander, United States Navy
B.S., University of Washington, 1958

Allen Borden Webb
Major, United States Marine Corps
B.S., University of Missouri, 1963

James Richardson King
Lieutenant, United States Naval Reserve
B.S., Louisiana State University, 1966

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the
NAVAL POSTGRADUATE SCHOOL
December 1972

ABSTRACT

An IBM System/360 based interpreter program for the AN/UYK-7 multiprocessing computer system is presented. The program provides interpretive execution of the AN/UYK-7 instruction set, an interrupt-handling capability, a variable-sized simulated memory and a multiprocessing capability for up to three central processors. Input/output is limited to a card reader and a line printer. Various segments of the program are discussed and a detailed User's Manual is provided.

TABLE OF CONTENTS

I. INTRODUCTION

In the mid 1960's the U. S. Navy contracted with the Univac Division of the Sperry Rand Corporation for a reliable, ruggedized, multiple-processor computer system which would meet the stringent environmental and functional specifications for ship and shore requirements of the Navy. As a result, Univac produced the AN/UYK-7 which the U. S. Navy has purchased in quantity for NTDS shipboard applications.

Since many of the naval officers who will work with the AN/UYK-7 will be Computer Science graduates of the Naval Postgraduate School it was felt that some exposure to the AN/UYK-7 system would be beneficial. For lack of the hardware, an interpreter was desired as an educational aid.

The remainder of this thesis will not be concerned so much with how an actual AN/UYK-7 works as it will be concerned with how the interpreter has been implemented and how to use the interpreter as an educational aid. Some shortcomings of the implementation and areas which may be the subject of future research will also be presented.

The following sections are primarily designed as an explanation and guide for the user. It is assumed that the user is thoroughly familiar with [Refs. 1, 2]. For the casual reader who is more interested in a general overview, or not familiar with the AN/UYK-7, it is suggested that the detail in section III-C-5 and section IV (User's Manual) be bypassed.

Some of the particular features incorporated in the interpreter are:

1. Variable-sized simulated AN/UYK-7 memory;

2. Multiprocessing;

3. Interrupt-handling capability.

The variable-sized simulated memory is covered in sections III-F (Memory and Paging) and IV-B (Job Control Language). Multiprocessing is covered in section III-B. Interrupt handling is covered in section III-D.

A minimum of 52K bytes of 360 memory is required to run any AN/UYK-7 configuration, however, with this minimum core, execution time may become unacceptable due to the paging demands.

II. DESCRIPTION OF THE AN/UYK-7

The AN/UYK-7 is a general-purpose, digital data computer with a multiprocessing capability that allows up to three central processors to operate simultaneously but independent of one another. Communication between the central processor (CP), memory units and I/O controller (IOC) is via a bused communication system consisting of three types of buses: instruction, operand and I/O. Each CP interfaces with the memory units via both instruction and operand data buses and with an IOC via an operand data bus. Every IOC communicates with the memory units via an I/O bus. Each CP is capable of addressing up to 16 memory units, a memory unit contains 16,384 words, 32 bits in length, and is capable of controlling up to four IOC's.

An IOC is capable of addressing up to 16 memory units and capable of communicating with external devices through an I/O adapter. The IOC can be controlled by a maximum of three different CP's. The I/O adapter is capable of communication with external devices via input and output channels that are provided in multiples of four. Each IOC has 4, 8, 12 or 16 I/O channels.

Each memory unit is capable of being addressed by any combination of CP's and IOC's, with the restriction that the total combined number of two accesses per CP and one access per IOC does not exceed eight memory accesses to any one memory unit.

Interface timing between computer units is handled on an asynchronous basis, wherein data transfers are performed via a request/acknowledge system. Each unit contains its own separate timing sources.

The computer contains three types of memory: main memory, control memory and non-destructive read out (NDRO) memory. Main memory is destructive read out memory consisting of three 16,384-word units. Maximum expansion capability is 16 units (262,144 words). This memory stores all instructions and operands processed during normal programmed operation. The control memory consists of 82 flip-flop storage registers (CMR's) that hold up to 32 bits each. The control memory, located in the CP, stores data used for operand address modification, interrupt execution and monitor clock update operations. The NDRO consists of 512 hard-wired words.

The NDRO memory is contained in the CP and is a predetermined set of programs that are fabricated as a permanent part of the CP. The purpose of the NDRO memory is to provide a means of automatically loading control memory and main memory, provide a hardware fault analysis program and a load-failure analysis program.

Computer operations can be controlled from three different sources: the operator panel, the maintenance console and the remote console. The operator panel initiates and controls normal on-line computer operation under program control. The optional maintenance console is used for off-line maintenance purposes. The optional remote console initiates

and monitors computer operations from a remote location. The maintenance and remote consoles are not required for system operation.

Execution of instructions by the CP is carried out in one of two modes, either the Interrupt or Task mode. The repertoire of the CP consists of 131 basic whole-word and half-word instructions of which 16 are privileged instructions and are executable only in the Interrupt mode. There is a separate set of 27 instructions provided for the IOC.

The AN/UYK-7 accomplishes arithmetic in ones-complement form. Fixed point binary and floating point binary arithmetic are available. Fixed point arithmetic can manipulate either 32 or 64 bit operands. Floating point operands consist of two 32 bit fields which contain a sign extended 15 bit characteristic and a signed 32 bit mantissa.

III. IMPLEMENTATION OF THE INTERPRETER

References 1 and 2 were the basic reference manuals used in the implementation. Disparities existed in both manuals concerning particular actions taken by the CP, and they were resolved by either background information or from outside sources. The outside sources were the Compiler Support Group and the SHARE-7 Group of the Fleet Combat Direction Systems Support Activity, San Diego, California.

A. PL360 LANGUAGE

PL360 is a language that provides all the facilities provided by System/360 Assembler Language yet exhibits an ALGOL-like syntax and control facilities. It was designed to improve the readability of programs written to take advantage of specific capabilities and limitations of the System/360.

PL360 was the language chosen over other suitable languages (e.g., PL/1) primiarily because of the efficient code produced. Reference 3 contains a formal description of the language. The version used was the O/S - DOS version of December 14, 1971.

Since PL360 does not provide interface routines, a subroutine is contained in Appendix A which provides an I/O interface for a card reader, punch and line printer. Appendix B contains the subroutines necessary to accomplish page file management.

B. MULTIPROCESSOR LOOP

The Multiprocessor Loop (MPL) is entered after each instruction is completed or after the INTERRUPT procedure has completed its switching of registers. The MPL updates the IOC Monitor Clock and the Real-Time Clock. The IOC Monitor Clock interrupt is generated in the MPL. The IOC Monitor Clock and the Interprocessor interrupts are considered by the MPL and the active CP or designated CP respectively will process these interrupts. The MPL determines the CP to run next in a case statement using a variable for each CP called CPUSTATE to determine the status of the CP being considered. The possible CP states are:

1. Halted;
2. Wait (waiting for interrupt);
3. Interprocessor Interrupt;
4. Active;
5. IOC Monitor Clock interrupt generated.

Only functions that will generate states three or five will remove a CP from the wait state.

If a configuration with more than one CP is selected (section II-C) the CP's will execute one instruction each in a round-robin fashion beginning with CP zero. If a CP becomes inactive it will still be considered in the MPL but it will not execute instructions.

C. CENTRAL PROCESSOR AND BASIC INSTRUCTION SET

1. Central Processor

Each CP has 82 Control Memory Registers (CMR's) which contain all the accumulators, index and base registers for both states of the CP plus the necessary registers to perform storage protection. Because many of the registers in the CMR are of different length, all CMR's were implemented as full 32-bit registers (see Table 1). Two additional registers were added to the CMR's. They are hereafter referred to as SIMTIME and MAINSWITCH. SIMTIME contains the total execution time for the CP. For further details on timing see section III-G. MAINSWITCH was incorporated to allow the user to simulate the setting of various switches on the maintenance console which directly affect the results of an instruction. The switches are Jump 1, 2 and 3, Breakpoint, Stop 5, 6 and 7. To use them see section III-C. For initialization see section IV-E.

Each CP has its own set of CMR's and there is no communication or transfer of information between them. All CMR's are initialized to zero except the following:

1. Active Status Register (ASR) is set in a Class IV interrupt state;
2. CP Monitor Clock has a value of negative zero.

To set the contents of the CMR prior to user program execution see section IV-E.

Although the CMR's were implemented as 32-bit values, in very few cases may the user take advantage of this as the interpreter only uses those bits required by an instruction. Two points of the implementation are not explicitly explained in Ref. 1. The first being that ASR bits 10 and 11 determine which set of base registers, accumulators and index registers are used in processing an instruction,

not the state of the CP. For example, the CP is in Task state and bit 10 in the ASR is set and bit 11 is off. The instruction would use the Task set of base registers and the Interrupt set of accummulators and index registers. The second point concerns wrap around of the accumulators. If an instruction references accumulator A(a) and A(a+1) with the value of a being seven then accumulator A(a+1) is accumulator A(0).

The Interrupt state has been defined as one of the bits 16-19 of the ASR being set. Bit seven of the ASR has no effect because the NDRO memory was not implemented.

2. Program Flow

Whole-word pre-processing is contained in the main program. The instruction is broken down into fields, indirect addressing is checked for and performed and the final operand address is computed. Based upon the function code, one of several procedures is called to perform the instruction. If abnormal termination or an error occurs in the instruction, the procedure INTERRUPT is called then control returns to the MPL. After normal completion, the TIME procedure is called to update the necessary clocks and SIMTIME.

For half-word instructions, the procedure HALFWORD is called and all pre-processing of the instruction is performed there. Normally, two half-word instructions are executed before control returns to the MPL. Abnormal termination of a half-word instruction invokes the INTERRUPT procedure.

If the lower half-word instruction, bits 15 - 0, is all zeros, then no operation is performed. If bits 31 - 26 are all zeros then an illegal instruction interrupt occurs.

3. Indirect Addressing

Indirect addressing is done prior to instruction execution in the pre-processing of the instruction. If character addressing is to be done the procedure MEMORY has the necessary information to obtain those bits requested when the operand fetch is done. When sequential character addressing is done the same information is available to the procedure MEMORY but the Indirect Address Word is modified as required and is stored back into the proper location prior to complete instruction execution.

A field modification to the AN/UYK-7 was performed concerning character addressing and is not contained in Ref. 1. For those particular instructions which use the k designator for quarter-word, half-word and whold-word operand reads or stores with character addressing specified in the Indirect Address Word, the k designator is ignored and the operand is processed according to specifications in the Indirect Address Word.

If an instruction specifies character addressing and it is not allowed, a CP Illegal Instruction Error interrupt occurs.

4. Repeat Mode

The normal flow in the interpreter is the processing of one instruction then control returns to the MPL. The "Repeat" instruction (RI), function code 076, alters this flow. Upon execution of the RI, control returns to the whole-word pre-processing section. If the next instruction is not repeatable then a CP Illegal Instruction Error interrupt is generated.

For termination in the non-compare instructions with the k designator in the RI being either five or six the contents of A(a) in the repeatable instruction are checked for even or odd parity.

A field modification to the AN/UYK-7 not contained in Ref. 1 concerns character addressing when it is specified in the repeatable instruction. The instruction is performed one time and B(7) is set to zero. If sequential character addressing is specified, the Indirect Address Word is modified appropriately, stored away and the instruction is done one time with B(7) set to zero.

5. Instruction Format Description

The central processor instruction set is divided into five types– Format I, II, III, IVA and IVB. Format I, II and III are whole-word instruction, 32 bits in length. Format IVA and Format IVB are half-word instructions, each 16 bits in length. See Refs. 1 or 2 for field designators and details of the instructions.

a. Format I Instructions

Format I instructions cover function codes (hereafter referred to as opcodes) 10 through 47 and 54 through 57. Opcodes 10 through 47 utilize the k) designator to specify whether the operand is read from or stored into memory in whole-word, half-word or quarter-word form. When the k designator is interpreted as a normal read instruction (memory to arithmetic), the procedure NORMREAD performs this function. The procedure NORMSTORE handles the case when the k designator is interpreted as a normal store instruction (arithmetic to memory). If the k designator is interpreted as a normal replace instruction the procedure NORMREAD is used for the read portion and the procedure NORMSTORE is used for the store portion.

For the normal store and normal replace instructions with the k designator equal to zero, a no operation instruction occurs. If character addressing is done (see section II-C-3) the k designator is ignored and character addressing is performed as required by the instruction.

For example, opcode 24 stores A(a) into the memory cell specified by the computer operand address. For k not equal to zero and no character add addressing, one of seven cases of NORMSTORE will be done; with k equal to zero and no character addressing, a no operation instruction is performed. If character addressing is specified, the accumulator is stored according to the specification of the character addressing, regardless of whether k is zero or not.

Opcodes 11 and 25 with the b designator equal to zero or opcodes 20 through 23 and 43 with the a designator equal to zero are executed as no-operation instructions. Opcodes 31, 32 and 42 are not replace instructions even though a disparity exists in Ref. 1 concerning this point. If the ak value is greater than 40, a no-operation instruction is done.

Opcodes 54 through 57 do not use the NORMREAD or NORMSTORE procedures. The k designator is special and is used as specified. As explained earlier, all CMR's are implemented as 32-bit fields. When executing opcodes 54 and 55, all 32 bits of the contents of Y are loaded execpt when CMR address 6X is used. The lower 15 bits are loaded and the upper bits are not affected. Opcode 56 and 57 store the full 32 bits into Y. For all four instructions, if CMR addresses 30 through 57 or 130 through 137 are accessed, a CP Illegal Instruction Error interrupt is generated. When a CMR is accessible in the Interrupt mode only, i.e., addresses 20 through 177, and the CP is in the Task mode, a Privileged Instuction Error interrupt is generated. For the rest of the Format I instructions no restrictions are placed upon the user except as noted in Ref. 1.

b. Format II Instructions

All fields are used in accordance with Ref. 1.

Opcode 06, with f2 equal to zero, one, two or three, is implemented utilizing bit manipulations and integer arithmetic due to the fact the 360 floating point hardware will not handle values acceptable in the AN/UYK-7. The same restrictions as they apply in Ref. 1, are required in the use of these opcodes.

Opcode 06, with f2 equal to four, five, six or seven, is not implemented and will return the same results as the corresponding opcode 06 instruction with no round.

Opcode 07, with f2 equal to one, two three or four ignores the designator because only one IOC was implemented. Opcode 07, with f2 equal to four, invokes the IOC procedure. In opcode 07, with f2 equal to five, the lower 20 bits of the Active Status Register (ASR) are replaced. This is the only place where bits 16 - 19 of the ASR may be changed by software other than in the INITIALIZE procedure. In other words, by altering the appropriate Designator Storage Word in the CMR, an ASR may be built to the user requirements. See section III-C-4 for an explanation of opcode 07 with f2 equal to six.

c. Format III Instructions

For all opcodes, the k designator, bit 20, must be zero. If not, a CP Illegal Instruction Error interrupt is generated. In opcode 53, with f3 equal to two or three and the a designator greater than zero, a variable called MAINSWITCH is accessed to see if the switch is set (see section II-C-1). These switches are set during the initialization phase (see section IV-E) and can not be altered once execution has begun. If the stop switches are selected and are on, the particular CP executing that instruction halts and can not be restarted.

d. Format IVA Instructions

In opcodes 60 and 61, 32 bits are transferred except for the following CMR addresses:

1. For CMR 11 through 17 and 111 through 117, only the lower 16 bits are moved;
2. For CMR 7X, in opcode 61, only the lower 15 bits are moved, the upper bits remain the same.

Opcode 74, with f4 equal to two, was implemented by a binary search and first approximation method since available square root routines would not handle a 64-bit integer. For opcode 74, f4 equal to three and the b designator equal to zero, the index register specified by the a designator is zeroed out. In opcode 74, f4 equal to seven, two special cases occurred. If the b designator is zero then B(a) is compared with zero, and/or if the a designator is zero then B(b) is compared with zero. The a designator in opcode 77, f4 equal to zero or one, is ignored since there is only one IOC. For opcode 77, f4 equal to six and the i designator is zero, the CP executing this instruction halts and can not be restarted. With the f4 designator equal to one, the only interrupts that will change the state of the CP are in Interprocessor interrupt or an IOC Monitor Clock interrupt, provided this particular CP receives the IOC Monitor Clock interrupt.

e. Format IVB Instructions

The procedure SHIFTAMT, interprets the m designator and performs the proper shift. If the shift amount is contained in B(b) and the b designator is zero, a no operation instruction occurs.

D. INTERRUPTS

The INTERRUPT procedure is implemented to set up the CP for all interrupts of Class II, III, IV and limited Class I interrupts (addressing

locations outside of the physical memory of the AN/UYK-7). Since the NDRO is not implemented, all interrupts load the branch address from the appropriate Initial Condition Word into the P register. Interrupts are generated when detected and INTERRUPT is called immediately with register one containing the class and register two containing the Interrupt Status Code. Interrupts are not queued due to the serial nature of the interpreter. The CP presently active (last executing) will process the interrupt generated with the exeception of the two interrupts considered in the MPL.

In INTERRUPT the p register and the old Active Status Register (ASR) are stored in the control memory of the CP and the ASR is changed as appropriate for the class of interrupt (Ref. 1 or Ref. 2).

E. INPUT-OUTPUT CONTROLLER

The IOC was implemented to closely simulate the actual IOC even though in a few instances reductions could have been made in the coding at the expense of realism. When the CP calls the IOC to do an instruction or a series of instructions (chaining) the IOC maintains control until it completes its instructions or an interrupt is generated in the IOC. This implementation of completing a series of IOC instructions before control is returned to a CP prevents the use of (or the requirement for using) a'Wait for Interrupt' instruction by the CP to ensure the buffer area is filled or emptied before processing can continue. When an interrupt is generated, control is returned through INTERRUPT to the MPL to process the interrupt and any instructions remaining in the IOC chain are not executed. This is one main difference from the actual AN/UYK-7 IOC. Buffer instructions are terminated by filling or emptying the buffer since the 'Terminate' instruction could not be efficiently implemented. Other

instructions not included in the IOC are the 'External Interrupt' and 'External Function' instructions since devices were not available to simulate these types of data transfers. The k designator in the IOC instruction field is implemented only to indicate data suppression or data transfer and not half-word or byte transfers as in the AN/UYK-7 (Refs. 1 and 2).

Two channels have been included in the IOC. Channel five is the card reader and channel six is the line printer. Input and output to these devices will be in EBCDIC format to run on the IBM-360. Capability to include other I/O devices on other channels requires modifying the case statement for each channel number.

F. MEMORY AND PAGING

The memory of the AN/UYK-7 is simulated in multiples of 1024 word modules or pages. An actual AN/UYK-7 would have some multiple of 16,384 words of memory, each word containing 32 bits. However, 16,384 words of AN/UYK-7 memory are the equivalent of 65,536 bytes of 360 memory. In order to reduce memory requirements, the interpreter program allows memory requirements to be defined in multiples of 1024 words or 4096 bytes. A user supplied control card defines the number of pages the job will require. Whether or not paging will be required is then determined internally by the program. By performing a shift left of 12 bits on the number of modules requested, the program determines the number of bytes required for the AN/UYK-7 memory. If the region declared for the GO step is adequate to allow the requested number of bytes to be obtained through the execution of a variable GETMAIN macro then the AN/UYK-7 memory will be wholly contained in core within the program. A logical flag called PAGING will remain false.

If the number of pages of AN/UYK-7 memory exceeds the region obtained by the GETMAIN macro then PAGING is flagged as true and the page file is created on disk. This occurs within procedure INITIALIZE. The page file is created as a Basic Direct Access file. When paging will be required, the user should ensure that the GO.PAGEFILE JCL card (see section IV-B) specifies at least as many records as the number of pages requested or the program will terminate during file initialization due to lack of file space on disk.

Procedure MEMORY contains the programming which performs all AN/UYK-7 memory references. MEMORY contains internal procedures FETCHPAGE, MEMINTERRUPT, MEMFETCH and MEMSTORE. MEMFETCH performs all AN/UYK-7 memory fetch references while MEMSTORE performs all AN/UYK-7 memory store references. When paging is in effect, either MEMFETCH or MEMSTORE may call FETCHPAGE due to a virtual page not being core resident. MEMINTERRUPT is called either by MEMFETCH or MEMSTORE whenever a requested AN/UYK-7 memory reference is outside of the defined limits of memory as described by the control card statement. MEMINTERRUPT then sets up the parameters and calls procedure INTERRUPT. All Class I interrupts that can be handled in software are generated in this manner.

Procedure FETCHPAGE contains the page replacement algorithm. The array PAGETABLE is scanned to a depth equal to the number of virtual pages if necessary, searching for a page which is described by one of the four cases which follow:

1. In memory, unchanged, not recently referenced;
2. In memory, unchanged, recently referenced;
3. In memory, changed, not recently referenced;
4. In memory, changed, recently referenced.

Given the worst situation case four will occur. If case one or two occurs then the page need not be written out. If case three or four occurs then the page must be written out and then replaced by the new page.

The foregoing cases are encoded in a status byte in the high-order byte of each virtual page entry in PAGETABLE. The eight bits (7-0) of the status byte are used in the following manner. Bit seven, the high-order bit, is on if the virtual page is core resident and off if not core resident. Bit six is on if the page has not had a store reference, otherwise bit six is off to indicate that the page has been modified. Bit five is on if the page has not been referenced since the last page fetch occured. This is due to the fact that all pages are changed to indicate an unreferenced condition when a new page is loaded into memory by PAGEFETCH. The remaining bits (4-0) of the status byte are unused.

Within each virutal page entry in PAGETABLE, the lower-order byte contains the physical page number (relative block address in memory) that the virtual page occupies or last occupied. When a virtual page is replaced, its physical page number is placed in the lower-order byte entry of the replacing page. Thus the physical page numbers are passed from virtual page to virtual page.

When paging is not required, the AN/UYK-7 address together with the starting address of the AN/UYK-7 memory area in core are used to directly compute the 360 address.

All calls to MEMORY pass a value in register one. The value in register one determines which type of memory request is being made via execution of a case statement. The first three cases are instruction fetch, operand fetch and operand store. For the operand store case, the value to be stored is assumed to be in register two. Each case carries out the appropriate storage protection checks if required. Cases four

and five are unconditional memory fetch and store cases used by the IOC since the IOC is not in any way restricted by storage protection. Case six is also an unconditional fetch case. This is due to normal store instructions which insert bytes into a word in AN/UYK-7 memory without fetching the word. Since MEMORY only fetches and stores whole-words a fetch must be made initially for the word in question in order to construct the final word. If the fetch were to violate storage protection in some way then an interrupt would be generated for a fetch violation by an instruction which supposedly does not fetch. Thus case six bypasses the problem since the following store operation will cause the proper interrupt if the storage protection conditions are appropriate. Case seven handles indirect addressing references. After storage protection checks for the indirect case are completed the program loops back to the case for operand fetching to complete the remaining storage protection checks.

If any call to MEMORY succeeds in passing the checks of the appropriate case or cases then ultimately a call will be made either to MEMFETCH or MEMSTORE. It is noteworthy that initialization and dump of AN/UYK-7 memory call MEMORY using cases four and five.

G. TIMING

1. AN/UYK-7 Clocks

All clocks indicate time in tenths of microseconds. The CP Monitor Clock initially set negative is updated after the completion of each instruction by the time required for the AN/UYK-7 to execute the instruction. SIMTIME is incremented by the same amount. The IOC Monitor Clock, initially set negative, and the Real-Time Clock, initially set zero, are updated in the MPL by an average time of two microseconds.

2. Speed Of The Interpreter

The sample program (see Computer Output) ran on the IBM-360 Model 67 with an average execution time ratio of 360 to 1. When memory and register dumps and trace were eliminated the sample program executed with a ratio of 240 to 1. It is noteworthy that the IOC time of the AN/UYK-7 is not included in SIMTIME from which these ratios were calculated. This implementation would provide accurate timing only if there was 100% overlap between CP execution and I/O and there was no interference between the CP's and the IOC.

IV. USER'S MANUAL

To utilize the interpreter the user must be familiar with the AN/UYK-7 instruction set and format and have a general understanding of the system configuration and requirements. The procedures INITIALIZE and INITIALIZ2 build a system for the user from data on control cards, allow presetting of the CMR's from data cards and allow loading of instructions and data into memory from instruction and data cards. The procedure INITIALIZE has very few error messages, therefore, when a field on the inputed card is either alphabetic or numeric, the user must assure that it follows the specification or a 360 system error may result.

A. INITIAL PROGRAM LOAD

CMR's 00 through 6X, 100 through 107, 111 through 177 and the Real-Time Clock have been initialized to zero. CMR 6X (ASR) has been initialized to a State IV interrupt condition. The CP Monitor Clock (CMR110) and the IOC Monitor Clock are initialized to negative values. The initialization of the CMR's applies to all three CP's. MAINSWITCH, the variable representing the switches on the maintenance console for each CP is initialized to zero. This means the Jump 1 - 3 and Stop 5 - 7 switches are off and the Breakpoint switch is set to the Program mode. SIMTIME is initialized to zero.

The zeroing of the CMR's has several important implications. First of all, since all of the Initial Condition Words for all of the CP's are preset to zero, all interrupts for any CP will cause instruction execution to branch to location zero in memory. Secondly, the Breakpoint CMR value

will not be checked because bits 18 and 19 will be off. Thirdly, initial execution in the Task state will immediately result in an interrupt because the storage protection registers are zeroed. Thus the user must insure that all applicable CMR's are properly set.

The CP Monitor Clock and the IOC Monitor Clock are set to negative values in order that the first instruction executed by a CP does not create an interrupt.

B. JOB CONTROL LANGUAGE

In Appendix C, there are two sets of Job Control Language (JCL) to cover the needs of the user. The reference to the AN/UYK-7 program in the JCL refers to data loaded into the interpreter and the PL360 program refers to the code for the interpreter.

If the user does not desire to build a load module on disk, the SYSLMOD card should be modified accordingly. In the first set of JCL there are two parameters which may be varied depending upon the size of AN/UYK-7 memory desired. They are the region parameter on the GO EXEC card and the space parameter on the GO.PAGEFILE card. See the next section for amplication.

C. CONTROL CARDS

There are six control cards and each one must be supplied or the program will terminate with an appropriate message. It should be stressed again that the formats for the cards be exactly followed or a 360 system error may result.

The first card specifies the number of 1K (words) segments of AN/UYK-7 memory. The number must be numeric and right-justified in columns one through three. If a value greater than 256 is requested, only 256 segments

will be allocated. The range of AN/UYK-7 memory address will be from zero to 1024 times the number of pages specified by the control card less one.

The interpreter and system routines require approximately 48K bytes of 360 core. The GO.PAGEFILE card will be required if not enough core is specified to contain the requested number of 1K segments (pages) of AN/UYK-7 memory. An easy method of determining whether the GO.PAGEFILE card is required is to take the region parameter specified on the GO EXEC card minus 48K and divide the result by 4K. The answer will be equal to the number of pages in core and if the number of pages requested exceeds the number in core the GO.PAGEFILE card is required with the space parameter as SPACE = (4096, number requested on control card).

Example 1: The user has allocated 100K for the GO step (i.e., REGION = 100K) and on the control card requests 13 segments of AN/UYK-7 memory. By the formula, (100K - 48K)/4K = 13, no paging is required. Therefore, the GO.PAGEFILE card is not required.

Example 2: The user has allocated 100K for the GO step and requests on the control card 20K (words) of AN/UYK-7 memory. Again, by the formula, the number of pages would be 13 and would be less than the number of pages requested in the control card. Therefore, paging will be required and the GO.PAGEFILE must have the space parameter specified as SPACE = (4096,20) or a 360 system error will occur.

The second control card specifies the number of CP's wanted. The number must be numeric and in column three. If the value is zero or greater than three, a message will be printed and the program terminated. CP numbers are allocated beginning with zero. Upon program execution those CP's allocated will be in an active state and will commence executing instructions.

The third control card specifies the sections of memory to be dumped at normal program termination. At least one and up to six different sections can be dumped. The card format is as follows:

1. Column 1 and 2 - "MD";
2. Columns 4 - 9, 17 - 22, 30 - 35, 43 - 48, 56 - 61 and 69 - 74 contain a six digit decimal number for the lower bound addresses;
3. Columns 10 - 15, 23 - 28, 36 - 41, 49 - 54, 62 - 67 and 75 - 80 contain a six digit decimal number for the upper bound addresses.

If any of the following conditions occur, that field and the remainder of the card are ignored:

1. Column 9, 15, 22, 28, 35, 41, 48, 54, 61, 67, 74 or 80 is blank;
2. The lower bound address is greater than the maximum memory address;
3. The upper bound address is greater than the maximum memory address;
4. The lower bound address is greater than the upper bound address.

The fourth card specifies which Control Memory Registers (CMR's) are to be dumped at normal program termination plus options concerning instruction input. There are 177 (octal) CMR's plus the P register, SIMTIME and MAINSWITCH or 202 (octal) registers. Any one consecutive section of the 202 registers may be dumped. The format of the card is as follows:

1. Columns 1 and 2 have "RD";
2. Columns four through six contain the lower octal CMR address (must be zero or greater);
3. Columns eight through ten contain the upper CMR address (must be zero or greater and less than 203).

If the upper address is less than the lower address a partial dump will occur. The two fields must be octal values, right justified in the proper columns with no blanks and supplied.

The next three fields on the register dump card concern the instruction input. In columns 12 and 13 a "PI" will print the cards with octal instruction format along with their load address. If "PL" is put in columns 15 and 16 then the data values inputed are echo-printed out. Columns 18 and 19 specify which instruction input is to be used. "F1" is

used for octal instruction format and "F2" is used for object instruction format. The defaults for the three fields are no instruction print, no data print and octal instruction format.

The fifth card allows the user to trace instructions in execution. The trace is printed when the address for the instruction falls within the trace card limits. Instructions executed by the 'Execute Remote' instructions are not traced.

This card is scanned based upon the number of CP's specified, i.e., if three CP's are requested then three fields (for CP zero, CP one and CP two in order) are scanned. The format of the card is as follows:

1. Column 1 contains a "T";
2. Column 2 contains a blank or "N", if "N" then no trace is provided;
3. Columns 3 - 8, 17 - 22 and 31 - 36 are a six digit decimal address for the lower bound on the trace;
4. Columns 10 - 15, 24 - 29 and 38 - 43 are a six digit decimal address for the upper bound on the trace.

If a trace is not wanted for a particular CP with two or three CP's requested then the upper bound address must be less than the lower address.

On the memory dump, register dump and trace control cards, all numbers must be right justified in their respective columns and contain no alphabetic characters. The only numeric fields which are not decimal are two fields on the register dump card. The last card has a "%" in column one to signify the end of the control cards.

D. INSTRUCTION INPUT

The two formats for inputing instructions are octal and object. The octal format closely represents the AN/UYK-7 format for instructions and the object format is a binary output from an assembler or compiler.

The octal format has eight fields for whole-word instructions and follows the format of Format I instructions. Columns one through five

on the card contain the first five octal numbers of the instruction and column six has a zero or one to indicate the indirect address designator off or on respectively. Column seven contains the octal base register number and columns eight through eleven contain a decimal number less than 8192 for the basic operand address. For example, if the number 53423161238 was contained on the card in columns one through eleven it would be interpreted as function code 53, a designator equal to four, f3 designator equal to one, k designator of zero, b designator equal to three , i designator on, s designator of six and a basic operand address of 1238.

Half-word instructions have 12 fields and follow the format of Format IVA instructions. Fields one through five and seven through eleven are the first five octal fields, respectively, of two half-word instructions. Fields six and twelve are a zero or one bit for the i designator. If the half-word instruction is a Format IVB instruction the m designator is in fields four through six or fields ten through twelve. An example of two half-word instructions is 716230627450 and it would be placed in columns one through twelve. The interpretation of the upper half-word would be a function code 71, a designator equal to six, an f4 designator of two, the designator equal to three and the i designator off. For the lower half-word, the instruction would be interpreted as function code 62, the a designator equal to seven and the m designator equal to 450 which means the shift amount is in B(5).

Instructions are loaded sequentially beginning at memory address zero unless a decimal address is specified in columns 15 - 20 of the card. If an address is specified then the instructions are loaded sequentially from that address. Consecutive instructions do not require an address field. Any time an address exceeds the maximum address available in memory and error message is printed and execution is terminated.

Columns 21 - 80 are available for comments. Only the three low-order bits are inspected when an octal field is specified and for binary fields, the low-order bit is inspected. The two decimal fields on the card are the basic operand address and the load address and must contain only decimal digits or a 360 system error will occur. See the Computer Output for examples of octal instructions.

The object format does not perform any card conversion as all eight bits of a single column are used. The first three columns represent an 18-bit address where the first instruction on the card is to be loaded into memory. The next column contains the number of four character fields to be placed into memory. A number of 1 through 19 is required and the value is used to scan the card. Columns 4 - 80 contain 19 four character fields in which all 32 bits are loaded into memory.

The last card for the object format inputs the P register values for the number of CP's specified by the CP control card. A "P" must be placed in column 4. Columns 5 - 8 are for the P register value for CP zero, columns 9 - 12 are for the P register value for CP one and columns 13 - 16 are for the P register value for CP two. Only the lower 20 bits are used of the 32 bits inputed.

The "PI" option explained earlier echo-prints the octal format and not the object format. The last card after either the octal format instructions or the object format instructions is a card with "%" in column one.

E. DATA AND REGISTER INPUT

To allow the user complete flexibility in specifying the initial state of the AN/UYK-7, the ability to input values into memory or override the

initialized values of the registers, i.e., the 177 CMR's, P register, SIMTIME and MAINSWITCH, was incorporated.

To load data into memory the card format is as follows:

1. "D" in column one;
2. A six digit decimal number less than the maximum memory address in columns three through eight;
3. The hexidecimal value to be loaded in columns 10 - 17.

To load the registers the card format is as follows:

1. "R" in column one;
2. A six digit decimal number from Table 1 in columns three through eight;
3. The hexidecimal value to be loaded in columns 10 - 17.

To load the registers for CP one, add 85 to the number obtained from Table 1 and for CP two, add 170 to the number obtained from Table 1.

MAINSWITCH is comprised of seven switches as follows:

1. Jump 1 in card column 10;
2. Jump 2 in card column 11;
3. Jump 3 in card column 12;
4. Breakpoint in card column 13;
5. Stop 5 in card column 14;
6. Stop 6 in card column 15;
7. Stop 7 in card column 16.

To turn on a switch or to set Breakpoint to Manual mode, a "1" is punched in the indicated column. A zero sets the switch off or the Breakpoint to Program mode. For example, 01010000 will set the Jump 2 switch on and the Break point to Manual mode.

When the ASR is loaded only the lower 20 bits are placed in that register to preserve the CP number.

F. SPECIAL NOTES

After a proper initialization sequence the CP and memory configuration is printed out. Then the message, "START EXECUTION," is printed and execution of the AN/UYK-7 instructions begin. The message, "END EXECUTION," is printed if and only if normal termination of the program occurs.

It is strongly advisable to load a 'Halt' or 'Wait' instruction at location zero or provide an interrupt handler for unexpected AN/UYK-7 interrupts. An OC7 System/360 error may occur as a result of improper data, instruction or control cards.

G. EXAMPLE PROGRAM

The computer output contains an example of an AN/UYK-7 program executed by the interpreter. Following the header line, "OCTAL INSTRUCTION FORMAT," is a program listing which is an echo-print of the instruction cards preceded by their decimal load points.

In the example program, CPO executes the code between and including locations 100 and 155 while CP1 executes the code between and including locations 1030 and 1080.

CPO has the task of reading in data cards, echo-printing the input, converting the EBCDIC code to a somewhat modified unpacked decimal format and storing the result in an array starting at location 2100. When a number is stored in the array, location 2048 receives the updated address pointing to the last location in the array. When CPO reads a blank card signifying end of data, a flag at location 2049 is set to zero. CPO then executes a 'Wait for Interrupt' instruction. In the meantime, CP1 starts testing location 2048 to determine if there are at least two values in the array. As soon as there are two values in the array, CP1 begins a modified ripple sort. Each time, prior to fetching a number from a deeper position in the array, CP1 tests location 2048 to ensure that the deepest position in the array has not been reached. If the deepest position in the array has been reached then location 2049 is tested in order to determine whether or not the array is complete.

When the sorting has been completed, CP1 enters the Interrupt state, sends an Interprocessor interrupt to CPO and then halts. This action restarts CPO which then reconstructs the number into EBCDIC format (with leading zeros suppressed), prints out the array and then halts.

As may be seen from the output, a list of the input data follows the "START EXECUTION" header line. The result of the program may be seen following the example trace of instructions.

V CONCLUSIONS

The benificial aspects of the interpreter as an educational aid for the student are twofold. First, the interpreter may be used as an educational aid to demonstrate and teach the principles involved in building operating systems and interrupt handlers. The principles of multiprocessing and multiprogramming may be taught by utilizing software without the requirement for the necessary hardware. At the present time, the Naval Postgraduate School does not have readily available a suitable software teaching aid for operating system principles. The AN/UYK-7 interpreter, possibly modified to run under a time-sharing system, would provide this capability. Secondly, the interpreter will give the student the ability to investigate and evaluate problem areas of the Navy's command and control systems. It will also provide a means of familiarization with the AN/UYK-7 for future "Fleet" users.

To the researcher, the interpreter is a tool for evaluating or improving existing operating systems for the AN/UYK-7. New concepts for subsystems for the Navy's command and control systems may now be devised and tested without the previously necessary hardware.

A brief review of present day interpreters reveals three important facts. First of all, most interpreters do not provide I/O on the simulated computer, and second, available memory in the simulated computer is often restricted. Finally, multiprocessing generally cannot be performed in a manner adquate to test and evaluate multiprocessing systems. The AN/UYK-7 interpreter provides limited I/O capability. However, this limitation may be overcome by incorporation of external I/O routines or implementation of additional IOC channels (see Appendix D for amplification).

The variable AN/UYK-7 memory realized in the interpreter allows the user a spectrum of options. These options vary from a minimum core requirement of 52K bytes, with the associated paging overhead, to maximum available core, with reduced or eliminated paging overhead.

The use of the PL360 language versus several other production type higher-level languages has afforded the user a reduced execution time per AN/UYK-7 instruction. The execution speed ratio is considerably lower than that in many other interpreters.

Due to insufficient time and the various permutations and combinations of instructions and data, it was not possible to completely test all instructions.

Areas of future study for using or improving the AN/UYK-7 interpreter are listed as follows:

1. Implemention of an operating system including a "Bootstrap" for the interpreter;
2. Investigation of operating systems for "Fleet" use on the AN/UYK-7;
3. Implementation under a time-sharing system (with the ability to display/modify registers and memory and restart the program);
4. Investigation of application programs limited only by the I/O devices available;
5. Implementation of additional IOC channels to provide better I/O capabilities;
6. Implementation of an assembler for use with the interpreter.

CONTROL MEMORY ADDRESS TABLE

CMR ADDRESS		DESCRIPTION OF	
(OCTAL)	(DECIMAL)	CONTROL MEMORY ADDRESS	LENGTH
0 - 7	0 - 7	TASK ACCUMULATORS	32 EA
10	8	UNASSIGNED (ADDRESSABLE)	19
11 - 17	9 - 15	TASK INDEX REGISTERS	20 EA
30 - 57	------	UNASSIGNED (NOT USABLE)	-----
60 - 67	24	BREAKPOINT REGISTER	20
70 - 77	25	ACTIVE STATUS REGISTER	23
100 - 107	26 - 33	INTERRUPT ACCUMULATORS	32 EA
110	34	CP MONITOR CLOCK	19
111 - 117	35 - 41	INTERRUPT INDEX REGISTERS	20 EA
120 - 127	42 - 49	INTERRUPT BASE REGISTERS	18 EA
130 - 137	-------	UNASSIGNED (NOT USABLE)	-----
140 - 157	50 - 65	DSW AND ICW REGISTERS	20 EA
160 - 167	66 - 73	STORAGE PROTECTION REGISTERS	21 EA
170 - 177	72 - 81	STORAGE IDENTIFICATION REGS	21 EA
200	82	P REGISTER	20
201	83	SIMTIME	32
202	84	MAINSWITCH	32

Table 1

APPENDIX A

```
************************************************************************
*           OS/360 OPERATING SYSTEM INTERFACE FOR PL360                *
************************************************************************

         ICTL  1,71,18
         SPACE
***********************************************************************
***********************************************************************
         SPACE
         MACRO
&EP      ENTER
         ENTRY &EP
         USING &EP,15
&EP      STM   12,2,SAVE               SAVE REGISTERS
         L     12,=A($PL360IO)         ESTABLISH ADDRESSING
         USING $PL360IO,12
         DROP  15
         MEND
         SPACE
         MACRO
         EXIT
         LM    12,2,SAVE               RESTORE REGISTERS
         BR    14
         DROP  12
         MEND
         SPACE 2
$PL360IO CSECT
         SPACE
LINELEN  EQU   132                     PRINTER LINE LENGTH
LINESMAX EQU   60                      PRINTER LINES/PAGE
         PRINT NOGEN
         SPACE
*   GLOBAL PROCEDURE READ(R14)
*        (R0)  = BUFFER ADDRESS
*        (R13) = SAVE AREA ADDRESS
*        (R14) = RETURN ADDRESS
READ     ENTER
         TM    SYSIN+DOPEN,OPENMASK    TEST FOR OPEN DCB
         BO    READ1
         LR    2,0
         OPEN  (SYSIN,(INPUT))         ISSUE OPEN
         LR    0,2
READ1    CLI   EOF,X'FF'               TEST FOR PREVIOUS END-OF-FILE
         BE    ERRPROC
         GET   SYSIN,(0)               GET CARD
READ2    CLI   EOF,0                   SET CONDITION CODE
```

```
         EXIT
         SPACE
*                  EOD EXIT ROUTINE                                    *
         USING   $PL360IO,12
ENDRDR   MVI     EOF,X'FF'             EODAD EXIT
         B       READ2
         DROP    12
         SPACE   2
*   GLOBAL PROCEDURE WRITE(R14)
*        (RO)  = BUFFER ADDRESS
*        (R13) = SAVE AREA ADDRESS
*        (R14) = RETURN ADDRESS
WRITE    ENTER
         LR      2,0
         TM      SYSOUT+DOPEN,OPENMASK TEST FOR OPEN DATA SET
         BO      WRITE1
         OPEN    (SYSOUT,(OUTPUT))
WRITE1   PUT     SYSOUT,(1)            GET NEXT BUFFER ADDRESS IN R1
         MVC     0(1,1),CARRCONT       SUPPLY CONTROL CHARACTER
         CLI     CARRCONT,C'1'
         BNE     WRITE2
         MVI     LINECNT,0             RESET LINE COUNT AND SPACING
         MVI     CARRCONT,C' '
WRITE2   MVC     1(LINELEN,1),0(2)     TRANSFER BUFFER
         IC      2,LINECNT             INCREMENT LINE COUNT
         LA      2,1(,2)
         STC     2,LINECNT
         CLI     LINECNT,LINESMAX      TEST FOR FULL PAGE
         BL      WRITE3
         MVI     CARRCONT,C'1'         SET SKIP
WRITE3   LM      12,2,SAVE
         BR      14
         DROP    12
         SPACE   2
*   GLOBAL PROCEDURE PAGE(R14)
*        (R14) = RETURN ADDRESS
         ENTRY   PAGE
         USING   PAGE,15
PAGE     MVI     CARRCONT,C'1'         SET
         BR      14
         SPACE   2
*   GLOBAL PROCEDURE PUNCH(R14)
*        (RO)  = BUFFER ADDRESS
*        (R13) = SAVE AREA ADDRESS
*        (R14) = RETURN ADDRESS
PUNCH    ENTER
         TM      SYSPUNCH+DOPEN,OPENMASK
         BO      PUNCH1
```

```
         LR       2,0
         OPEN     (SYSPUNCH,(OUTPUT))
         LR       0,2
PUNCH1   PUT      SYSPUNCH,(0)            PUT CARD IMAGE
         EXIT
         SPACE    2
SYSOUT   DCB      DSORG=PS,MACRF=PL,DDNAME=SYSPRINT,DEVD=DA,RECFM=FBA,   X
                  LRECL=LINELEN+1,BFTEK=S,EROPT=ABE
SYSIN    DCB      DSORG=PS,MACRF=GM,DDNAME=SYSIN,DEVD=DA,RECFM=FB,       X
                  LRECL=80,BFTEK=S,EROPT=ABE,EODAD=ENDRDR
SYSPUNCH DCB      DSORG=PS,MACRF=PM,DDNAME=SYSPUNCH,DEVD=DA,RECFM=FB,    X
                  LRECL=80,BFTEK=S,EROPT=ABE
SAVE     DS       7F
ERRPROC  DC       H'0'
EOF      DC       X'00'
LINECNT  DC       X'00'                       PRINTER LINE COUNT
CARRCONT DC       C'1'                    PRINTER CARRIAGE CONTROL
         SPACE
         DCBD
$PL360IO CSECT
DOPEN    EQU      DCBOFLGS-IHADCB         BYTE SET BY OPEN
OPENMASK EQU      X'10'
         END
```

APPENDIX B

```
***********************************************************************
*        PROCEDURE GETFRAME - OBTAINS AN/UYK-7 MEMORY SPACE           *
***********************************************************************
         ICTL     1,71,18
         SPACE
************************************************************************
************************************************************************
         SPACE
         MACRO
&EP      ENTER
         GBLC  &CSECT
         ENTRY    &EP
         USING    &EP,15
&EP      STM      12,2,SAVE                 SAVE REGISTERS
         L     12,=A(&CSECT)                ESTABLISH ADDRESSING
         USING &CSECT,12
         MEND
         SPACE
         MACRO
         EXIT
         LM       12,2,SAVE                 RESTORE REGISTERS
         BR       14
         DROP     12
         MEND
         GBLC  &CSECT
&CSECT SETC '$FRAME'
         SPACE    2
&CSECT CSECT
*
*        GLOBAL PROCEDURE GETFRAME
*
*        FUNCTION:  OBTAIN STORAGE FOR PAGE FRAMES.
*              OBTAINS A SEGMENT OF CORE OF LENGTH (4K+SYSFREE)
*              TO THE MAXIMUM AMOUNT DESIRED BY THE USER.  RETURNS
*              TO CALLER WITH NUMBER OF FRAMES OBTAINED AND A POINTER
*              TO THE START OF THE AREA.
*
*        ENTRY: RO= MAX NUMBER OF PAGE FRAMES DESIRED
*
*        RETURN RO= NUMBER OF FRAMES ACTUALLY OBTAINED
*               R1= BASE ADDRESS OF PAGE FRAME AREA
*        NOTE: ROUTINE WILL FAIL IF THERE IS NOT ROOM FOR AT
*        LEAST ONE PAGE FRAME PLUS SYSTEM SPACE (SYSFREE).
*
         SPACE 5
SYSFREE  EQU   (4*1024)                     4K FOR BUFFERS, ETC.
```

```
COREMIN  EQU   (4*1024+SYSFREE)        MINIMUM ACCEPTABLE AMOUNT
         SPACE
GETFRAME ENTER
         SLL   0,12                    CONVERT PAGES TO BYTES
         A     0,=A(SYSFREE)
         ST    0,ASKLIST+4             SAVE MAX DESIRED AMOUNT
         GETMAIN VU,LA=ASKLIST,A=GOTLIST  GET MOST POSSIBLE IN LIMITS
         SPACE
         L     2,GOTLIST+4             SAVE AMOUNT OBTAINED
         LR    0,2
         S     0,=A(SYSFREE)           REDUCE BY SYSFREE AMOUNT
         N     0,=A(4095)              (TOTAL-SYSFREE) REM 4096
         A     0,=A(SYSFREE)           ADD SYSFREE AMOUNT
* ABOVE FEW LINES INSURE THAT WE KEEP ONLY WHOLE 4K CHUNKS
         ST    0,GOTLIST+4             SAVE AMOUNT TO FREE
         FREEMAIN V,A=GOTLIST
         SPACE
         L     1,GOTLIST               BASE ADDRESS BEFORE FREE
         A     1,GOTLIST+4   PLUS AMT FREED GIVES NEW BASE ADDR
         LR    0,2                     OLD TOTAL (BEFORE FREE)
         S     0,GOTLIST+4   MINUS AMOUNT FREED GIVES NEW TOTAL
         SRL   0,12          DIVIDED BY 4096 GIVES NUMBER OF PAGES
         STM   0,1,SAVE+16   STORE SO THEY GET RESTORED
         EXIT
ASKLIST  DC    A(COREMIN,0)
GOTLIST  DC    A(0,0)
SAVE   DC      7F'0'
         SPACE 5
         END


***********************************************************************
*              PROCEDURES INITPAGE - GETPAGE - PUTPAGE                *
***********************************************************************
         ICTL    1,71,18
         SPACE
************************************************************************
************************************************************************
         SPACE
         MACRO
&EP      ENTER
         GBLC  &CSECT
         ENTRY   &EP
         USING   &EP,15
&EP      STM     12,2,SAVE             SAVE REGISTERS
         L     12,=A(&CSECT)           ESTABLISH ADDRESSING
         USING &CSECT,12
         DROP    15
```

```
         MEND
         SPACE
         MACRO
         EXIT
         LM    12,2,SAVE                RESTORE REGISTERS
         BR    14
         DROP  12
         MEND
         GBLC  &CSECT
&CSECT SETC '$PAGE'
         SPACE 2
&CSECT CSECT
*        GLOBAL PROCEDURE INITPAGE
*
*        14=RETURN ADDRESS
*        13=SAVE AREA ADDRESS
*        0=POINTER TO BUFFER CONTAINING RECORD TO
*            BE WRITTEN.
*
*        NOTE: CALLING PROGRAM IS RESPONSIBLE FOR STOPPING BEFORE
*        FILE SIZE IS EXCEEDED (S B37 OR D37 ABEND).  FILE SIZE
*        AND BLOCKSIZE DETERMINED BY DD CARD.
*
         SPACE 3
INITPAGE ENTER
         LTR   2,0                    CLOSE??
         BM    INITCLOS                 YES
         TM    INITDCB+DCBOFLGS-IHADCB,X'10'  OPEN??
         BO    INITWRIT                 YES
         SPACE
         OPEN  (INITDCB,(OUTPUT))
         SPACE
INITWRIT WRITE INITDECB,SF,INITDCB,(2)
         CHECK INITDECB
         B     INITEXIT
         SPACE
INITCLOS CLOSE (INITDCB)
INITEXIT DS    0H
         EXIT
INITDCB  DCB   DDNAME=PAGEFILE,MACRF=(WL),DSORG=PS,RECFM=F,DEVD=DA
         SPACE 5
*
*        GLOBAL PROCEDURES GETPAGE AND PUTPAGE
*
*        14=RETURN ADDRESS
*        13=SAVE AREA ADDRESS
*        0=POINTER TO BUFFER
*        1=PAGE NUMBER (0 -> N-1)
```

```
*
         SPACE 3
GETPAGE  ENTER
         LTR    2,0                    CLOSE??
         BNM    GOPAGE                   NO
CLOSPAGE CLOSE  (PAGEDCB)
         B      PAGEEXIT                 RETURN
         SPACE
GOPAGE   ST     1,PAGENO                SAVE PAGENUMBER
         TM     PAGEDCB+DCBOFLGS-IHADCB,X'10'  OPEN??
         BO     READPAGE                 YES
         SPACE
         OPEN   (PAGEDCB,(UPDAT))
         SPACE
READPAGE READ   PDECB1,DI,PAGEDCB,(2),'S',0,PAGENO+1
         CHECK  PDECB1
         SPACE
PAGEEXIT DS     0H
         EXIT
         SPACE 5
PUTPAGE  ENTER
         LTR    2,0
         BM     CLOSPAGE                 CLOSE??
         ST     1,PAGENO
         TM     PAGEDCB+DCBOFLGS-IHADCB,X'10'  OPEN??
         BO     WRITPAGE                 YES
         SPACE
         OPEN   (PAGEDCB,(UPDAT))
         SPACE
WRITPAGE WRITE  PDECB2,DI,PAGEDCB,(2),'S',0,PAGENO+1
         CHECK  PDECB2
         B      PAGEEXIT
         SPACE 3
PAGEDCB  DCB  DSORG=DA,DDNAME=PAGEFILE,MACRF=(RIC,WIC),RECFM=F,OPTCD=R
PAGENO   DC     F'0'
SAVE     DS     7F
         DROP   12     MANUALLY DROP IT SINCE WE DON'T NEED "EXIT"
         DCBD
         END
```

APPENDIX C

JOB CONTROL LANGUAGE

```
***********************************************************************
*    JCL FOR EXECUTING AN AN/UYK-7 PROGRAM USING THE LOAD MODULE      *
***********************************************************************
  //JOBNAME JOB (XXXX,XXXXNT,XXX),'NAME'
  //JOBLIB   DD DSN=S2241.ANUYK7,DISP=SHR,UNIT=2314,VOL=SER=LINDA
  //GO   EXEC   PGM=INT,REGION=60K
  //GO.SYSPRINT  DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3325),
  //      SPACE=(CYL,(1,1))
  //SYSUDUMP   DD   SYSOUT=A
  //GO.PAGEFILE DD UNIT=SYSDA,SPACE=(4096,16),DCB=BLKSIZE=4096
  //GO.SYSIN   DD *
  (AN/UYK-7 PROGRAM DECK)
  /*


***********************************************************************
*JCL FOR CREATING A NEW LOAD MODULE AND EXECUTING AN AN/UYK-7 PROGRAM*
***********************************************************************
  //JOBNAME JOB (XXXX,XXXXNT,XXX),'NAME'
  //JOBLIB   DD   DSN=C0012.PL360,DISP=SHR,VOL=SER=MARY,UNIT=2314
  //COMP EXEC PGM=PL360,REGION=150K
  //SYSPRINT DD SYSOUT=A,SPACE=(TRK,(20,10)),
  //        DCB=(RECFM=FBA,LRECL=133,BLKSIZE=798)
  //SYSGO DD UNIT=SYSDA,SPACE=(TRK,(10,5),RLSE),DISP=(,PASS),
  //        DCB=(RECFM=FB,LRECL=80,BLKS9Z5=800'
  //SYSIN    DD *
  (PL360 PROGRAM DECK)
  //LINK       EXEC    PGM=IEWL,PARM='MAP,LIST',REGION=150K,COND=(0,NE,COMP)
  //SYSUT1     DD      UNIT=SYSDA,SPACE=(TRK,(20,10))
  //SYSLMOD DD DSN=S2241.ANUYK7(INT),UNIT=2314,VOL=SER=LINDA,
  //       DISP=(OLD,KEEP)
  //SYSLIB     DD      DSN=S0938.PLIO,UNIT=2314,VOL=SER=MARY,DISP=SHR
  //           DD   DSN=F1790.CSGROUP.LIBRARY,DISP=SHR,UNIT=2314,
  //       VOL=SER=LINDA
  //SYSPRINT   DD      SYSOUT=A,DCB=(RECFM=62A,395CL=121,BLKSIZE=1210),
  //         SPACE=(TRK,5)
  //SYSLIN     DD      DSN=*.COMP.SYSGO,DISP=(OLD,DELETE)
  //GO         EXEC    PGM=*.LINK.SYSLMOD,REGION=150K,COND=(4,LT,LINK)
  //SYSPRINT   DD      SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=798),
  //         SPACE=(TRK,(10,5))
  //SYSUDUMP DD SYSOUT=A,SPACE=(TRK,(10,5))
  //SYSIN DD *
  (AN/UYK-7 PROGRAM DECK)
  /*
```

APPENDIX D

EXTENDING I/O CAPABILITIES

The limitations imposed by the IOC would prevent running a generalized program such as the ULTRA-32 assembler or CMS-2 compiler which would require more complex I/O facilities. The following paragraphs describe two approaches to extend the capabilities of the IOC. It is important to note, however, that while it might be possible to run a large program such as a compiler, this is not the primary purpose for an interpreter and the execution time would be excessive.

The first alternative would be to implement the additional channels available in the IOC for other devices (e.g., tape, disk or punch). This method, while providing realism, requires considerable knowledge of PL360 and the AN/UYK-7.

A more efficient alternative than the first would be to use external I/O routines. These routines may be either IBM System/360 routines or routines written in compatible languages. The method of invoking these routines will require modification of the interpreter program. One method is to use the 'Enter Executive State' instruction, opcode 07 with f2 of 0, to call the external routines. One bit of the 16 bit field contained in this instruction could be a flag indicating that external I/O was the desired operation.

The recommended method is to modify the code for the 'Initiate I/O' instruction to include the use of the a field. This field then would indicate whether the IOC or the external I/O routines were to be invoked.

This second alternative would also eliminate the requirement that the user write IOC programs to execute the desired I/O.

Anyone modifying the interpreter program must consider the following possible consequence. The single data segment, which contains all declared variables, has only 640 bytes of additional area available. If declarations are increased above this amount data segment overflow will result and considerable modification of the interpreter program will be necessary.

COMPUTER OUTPUT

```
   CONTROL CARDS
  3
  2
MD 000100000106 000107000113 000114000120 000121000127 000128000134 000135000141
RD 000 201 PI  PL  F1
T 000120 000126 010000 001031

OCTAL INSTRUCTION FORMAT
000000   770600000000          HALT.  PREVENT LOOPING BY ANY CPU
000005   07100000001    000005 SEND INTER-PROCESSOR INTERRUPT TO CPU 0
000006   770600000000   000006 HALT CPU 1
000100   07040000700    000100 INITIATE I/O FOR READING A CARD
000101   07040000800           INITIATE I/O FOR WRITE TO ECHO PRINT THE CARD
000102   10130010501           LOAD A(1) WITH 2ND BUFFER WORD
000103   44130010300           COMPARE A(1) TO LOCATION 400
000104   53120010020           JUMP EQUAL TO SET FLAG AND WAIT FOR SORT
000105   10250010499           LOAD A(2) FROM Y(599) BITS 15-8,   A(2)=2
000106   10340010499           LOAD A(3) FROM Y(599)  BITS 7-0, A(3)=4
000107   650020641020          SHIFT A1, A0 RIGHT 4 BITS; SHIFT A1 RIGHT 4 BITS
000108   13370010499           DECREMENT A(3) BY 1
000109   44360010499           COMPARE A(3) TO ZERO
000110   53220010007           JUMP A(3) > 0 TO MEMORY LOCATION 107
000111   13270010499           DECREMENT A(2) BY 1
000112   44260010499           COMPARE A(2) TO ZERO
000113   53220010015           IF A(2) > 0 THEN GOTO MEMORY LOCATION 115
000114   51220010017           BRANCH TO STORE AT 117
000115   10130010500           LOAD A(1) WITH FIRST BUFFER WORD
000116   53060010006           BRANCH TO 106
000117   23210011948           STORE CONTENTS OF INDEX REG 2 IN LOCATION 2048
000118   25032000000           STORE A0 IN ARRAY BASED ON LOC IN INDEX REG 2
000119   51220010000           GOTO READ AT LOC 100
000120   702200000000          SET A(2) TO 0 BY COMPLIMENTING
000121   24230002049           SET FLAG AT 2049 TO ZERO TO INDICATE ARRAY COMPLETE
000122   770601000000          HALT WAIT FOR INTER-PROCESSOR INTERRUPT FROM CPU 1
000123   10700000000           LOAD A(7) WITH 0
000124   24730010500           LOAD PRINT BUFFER WITH 0
000125   24730010501           LOAD PRINT BUFFER WITH 0
000126   07040000800           INITIATE I/O (WRITE A BLANK LINE)
000127   10600000001           LOAD A(6) WITH A 1
000128   10540010499           LOAD A(5) FROM Y(599) BITS 7-0, A(5)=4
000129   10470010498           LOAD A(4) FROM Y(598) BITS 31-24, A(4)=8
000130   20210010498           LOAD INDEX REG 2 WITH ADDRESS OF ARRAY -1
000131   43210011948           COMPARE INDEX REG 2 TO LAST CELL LOC AND INCREMENT
000132   53620010055           IF CD OUTSIDE LIMITS BRANCH TO HALT AT Y(155)
```

```
000133   713330710300          ZERO OUT A(3) AND A(0)
000134   711310000000          ZERO OUT A(1)
000135   10232000000           LOAD A(2) FROM LOC SPECIFIED BY INDEX REG 2
000136   631020713160          SHIFT LEFT A1, A2 4 BITS THEN INCREMENT A3
000137   743440000000          COMPARE A(3) TO A(4)
000138   53120010051           JUMP EQUAL TO Y(151)
000139   741470000000          COMPARE A(1) TO A(7)
000140   53120010036           BRANCH EQUAL TO Y(136)
000141   01100010497           OR A(1) WITH #FO
000142   743450000000          COMPARE A(3) TO A(5)
000143   53120010046           BRANCH EQUAL TO Y(146)
000144   621020000000          SHIFT A(1) LEFT 4 BITS
000145   53060010036           MANUAL JUMP TO Y(136)
000146   630200000000          SWAP A(0), A(1) BY 32 BIT SHIFT CIRCULAR
000147   740470000000          COMPARE A(0) TO A(7)
000148   53120010036           JUMP EQUAL TO 136
000149   01100010496           OR A(1) WITH #OF
000150   53060010036           MANUAL BRANCH TO Y(136)
000151   01100010497           OR A(1) WITH #FO
000152   02070010500           LOAD A(0), A(1) INTO THE READ/WRITE BUFFER
000153   07040000800           INITIATE I/O (WRITE)
000154   53060010031           MANUAL BRANCH TO Y(131) TO GET NEXT WORD
000155   770600000000          HALT
002048   00000002099   002048  INITIALIZE LOCATION 2048 TO A VALUE OF 2099
002049   40000000000           INITIALIZE LOCATION 2049 TO A ONE IN SIGN POSITION
001030   07050010000   001030   ENTER TASK STATE
001031   10000002100            2100 -> A(0)
001032   24030011700            A(0) -> (1700)
001033   44030012048            A(0)=2100 : (2048)
001034   53320011033            IF A(0) >= GOTO 1033
001035   53060011060            UNCOND JUMP TO 1060
001060   20130011700   001060   (1700) -> B(1)
001061   10400000001            1 -> A(4)
001062   704200000000           A(4)' -> A(4)
001063   24430011701            A(4) -> (1701)
001064   23130011702            B(1) -> (1702)    START OF LOOP : LOAD
001065   10330011702            (1702) -> A(3)
001066   44330011700            A(3) : (1700)
001067   53020011069            JUMP IF ¬=
001068   35530011700            (1700) + 1 -> (1700) & A(5)
001069   05101010000            2100+(B(1)) -> A(1),2100+(B(1)+1) -> A(2)
001070   741420000000           A(1) : A(2)
001071   53320011080            JUMP IF > TO SWITCH
001072   20130011700            (1700) -> B(1)
001073   23130011704            B(1) -> 1704  LOOP
001074   10530011704            (1704) -> A(5)
001075   44530012048            A(5) : (2048)
001076   53420011064            JUMP IF < TO LOAD
```

```
001077  03070012049            TEST & SET (2049)
001078  53020011073            JUMP IF SET TO LOOP
001079  07000000000            ENTER EXEC STATE TO TERMINATE
001080  631200000000           SWAP A(1) & A(2) SWITCH
001081  02171010000            A(1) -> 2100+(B(1)),A(2)->2100+(B(1)+1)
001082  21130011701            B(1) - 1 -> B(1)
001083  23130011705            B(1) -> 1705
001084  44030011705            A(0) : (1705) = 2100 : B(1)
001085  53520011064            JUMP IF <= TO LOAD
001086  20130011700            (1700) -> B(1)
001087  53060011073            JUMP UNCOND TO LOOP
        %

REGISTER/DATA INITIALIZATION
D 000400 40404040
D 000596 0000000F
D 000597 000000F0
D 000598 08000833
D 000599 01000204     FIELD OF CONSTANTS USED BY PROGRAM
D 000602 02590258       BUFFER ADDRESSES FOR I/O INSTRUCTIONS
D 000700 2150025A           I/O READ INSTRUCTION
D 000800 2560025A       I/O WRITE INSTRUCTION
R 000110 00010F00
R 000147 00000005     LOAD ICW P REG VALUE CLASS IV INTERRUPT FOR CPU 1
R 000150 00020407
R 000151 FFFFFFFF
R 000152 FFFFFFFF
R 000167 00020406
R 000025 00110E00     SET UP ASR TO INDICATE USE OF INTERRUPT REGISTERS
R 000036 00000834     LOAD INDEX REG 2 WITH STARTING ADDRESS OF ARRAY
R 000043 00000064     LOAD BASE REGISTER 1 WITH 100
R 000054 0000007B    LOAD CLASS II INTERRUPT ADDRESS 123 INTO CMR 144(OCTAL)
R 000082 00000064     INITIALIZE P REG FOR CPU 0 TO LOCATION 100
%

INITIALIZATION PHASE COMPLETE

AN/UYK-7 CPU AND MEMORY CONFIGURATION

THE NUMBER OF CPU'S ACTIVE ARE 2
THE NUMBER OF PAGES IN CORE ARE 003
MEMORY AVAILABLE IS FROM 0 TO 003071
PROGRAM EXECUTION FOR CPU 0 WILL START AT LOCATION 000100
PROGRAM EXECUTION FOR CPU 1 WILL START AT LOCATION 001030
MEMORY DUMP IS FROM 000100 TO 000106
MEMORY DUMP IS FROM 000107 TO 000113
MEMORY DUMP IS FROM 000114 TO 000120
MEMORY DUMP IS FROM 000121 TO 000127
```

```
MEMORY DUMP IS FROM 000128 TO 000134
MEMORY DUMP IS FROM 000135 TO 000141
TRACE FOR CPU 0 IS FROM 000120 TO 000126
TRACE FOR CPU 1 IS FROM 010000 TO 001031

**** START EXECUTION ****

   23354
     109
       0
      12
     121
    2312
     122
    5432
      13
    5867
     110
     115
      14
     103
       2
     120
     118
       4
       0
     105
     102
 1234567
     108
   12115
   00022
     107
     123
     119
     104
     101
     106
       5
     112
     116
     100
     114
12345678
     113
     999
  123456
```

```
111
117

UYK-7 ADDRESS (DECIMAL): 000120  IN INTERRUPT STATE FOR CPU # 0
ACTIVE STATUS REGISTER (IN HEX): 00010E06  P REGISTER (IN HEX): 00020014
OP CODE: 70  ACCUMULATOR DESIGNATOR: 2  ACCUMULATOR A VALUE (IN HEX): FFFFFFFF
F4 DESIGNATOR: 2  INDEX DESIGNATOR: 0  I DESIGNATOR OFF
ACCUMULATOR A+1 VALUE (IN HEX): FFFFFFFF  A(B) VALUE (IN HEX): 00073758

UYK-7 ADDRESS (DECIMAL): 000121  IN INTERRUPT STATE FOR CPU # 0
ACTIVE STATUS REGISTER (IN HEX): 00010E06  P REGISTER (IN HEX): 00020015
OP CODE: 24  ACCUMULATOR DESIGNATOR: 2  ACCUMULATOR A VALUE (IN HEX): 00000000
INDEX DESIGNATOR: 0  NO INDEXING
NO INDIRECT ADDRESSBASE REG DESIGNATOR:BASE  0  BASE REG VALUE (DECIMAL): 000000
BASIC OPERAND ADDRESS (DECIMAL): 2049  FINAL COMPUTED ADDRESS (DECIMAL): 002049
K DESIGNATOR: 3  (Y) (IN HEX): 80000000
ACCUMULATOR A+1 VALUE (IN HEX): FFFFFFFF

UYK-7 ADDRESS (DECIMAL): 000122  IN INTERRUPT STATE FOR CPU # 0
ACTIVE STATUS REGISTER (IN HEX): 00010E06  P REGISTER (IN HEX): 00020016
OP CODE: 77  ACCUMULATOR DESIGNATOR: 0  ACCUMULATOR A VALUE (IN HEX): 00000117
F4 DESIGNATOR: 6  INDEX DESIGNATOR: 0  I DESIGNATOR ON

UYK-7 ADDRESS (DECIMAL): 000123  IN INTERRUPT STATE FOR CPU # 0
ACTIVE STATUS REGISTER (IN HEX): 00053E00  P REGISTER (IN HEX): 0000007B
OP CODE: 10  ACCUMULATOR DESIGNATOR: 7  ACCUMULATOR A VALUE (IN HEX): 00000000
INDEX DESIGNATOR: 0  NO INDEXING
NO INDIRECT ADDRESSBASE REG DESIGNATOR:BASE  0  BASE REG VALUE (DECIMAL): 000000
BASIC OPERAND ADDRESS (DECIMAL): 0000  FINAL COMPUTED ADDRESS (DECIMAL): 000000
K DESIGNATOR: 0  (Y) (IN HEX): FC600000
ACCUMULATOR A+1 VALUE (IN HEX): 00000117

UYK-7 ADDRESS (DECIMAL): 000124  IN INTERRUPT STATE FOR CPU # 0
ACTIVE STATUS REGISTER (IN HEX): 00053E00  P REGISTER (IN HEX): 0000007C
OP CODE: 24  ACCUMULATOR DESIGNATOR: 7  ACCUMULATOR A VALUE (IN HEX): 00000000
INDEX DESIGNATOR: 0  NO INDEXING
NO INDIRECT ADDRESSBASE REG DESIGNATOR:BASE  1  BASE REG VALUE (DECIMAL): 000100
BASIC OPERAND ADDRESS (DECIMAL): 0500  FINAL COMPUTED ADDRESS (DECIMAL): 000600
K DESIGNATOR: 3  (Y) (IN HEX): 40404040
ACCUMULATOR A+1 VALJE (IN HEX): 00000117

UYK-7 ADDRESS (DECIMAL): 000125  IN INTERRUPT STATE FOR CPU # 0
ACTIVE STATUS REGISTER (IN HEX): 00053E00  P REGISTER (IN HEX): 0000007D
OP CODE: 24  ACCUMULATOR DESIGNATOR: 7  ACCUMULATOR A VALUE (IN HEX): 00000000
INDEX DESIGNATOR: 0  NO INDEXING
NO INDIRECT ADDRESSBASE REG DESIGNATOR:BASE  1  BASE REG VALUE (DECIMAL): 000100
BASIC OPERAND ADDRESS (DECIMAL): 0501  FINAL COMPUTED ADDRESS (DECIMAL): 000601
```

```
K DESIGNATOR: 3  (Y) (IN HEX): 40404040
ACCUMULATOR A+1 VALUE (IN HEX): 00000117

UYK-7 ADDRESS (DECIMAL): 000126  IN INTERRUPT STATE FOR CPU # 0
ACTIVE STATUS REGISTER (IN HEX): 00053E00  P REGISTER (IN HEX): 0000007E
OP CODE: 07  ACCUMULATOR DESIGNATOR: 0  ACCUMULATOR A VALUE (IN HEX): 00000117
INDEX DESIGNATOR: 0  NO INDEXING
NO INDIRECT ADDRESSBASE REG DESIGNATOR:BASE  0  BASE REG VALUE (DECIMAL): 000000
BASIC OPERAND ADDRESS (DECIMAL): 0800  FINAL COMPUTED ADDRESS (DECIMAL): 000800
F2 DESIGNATOR: 4
ACCUMULATOR A+1 VALUE (IN HEX): 40404040

        0
        0
        2
        4
        5
       12
       13
       14
       22
      100
      101
      102
      103
      104
      105
      106
      107
      108
      109
      110
      111
      112
      113
      114
      115
      116
      117
      118
      119
      120
      121
      122
      123
      999
     2312
     5432
```

```
    5867
   12115
   23354
  123456
 1234567
12345678

ADDRESS   MEMORY DUMP
DECIMAL   HEXIDECIMAL

000100    1C4002BC 1C400320 20B021F5 90B0212C ACA02014 215021F3 21C021F3

000107    D404D084 2DF021F3 91E021F3 AD202007 2D7021F3 916021F3 AD20200F

000114    A5202011 20B021F4 AC602006 4D10279C 54340000 A5202000 E1200000

000121    51300801 FC610000 23800000 53B021F4 53B021F5 1C400320 23000001

000128    22C021F3 227021F2 411021F2 8D10279C AF202037 E5B6E430 E4B20000

000135    21340000 CC84E59C F1C80000 ACA02033 F0CE0000 ACA02024 048021F1

                 CPU NUMBER 0

          CONTROL MEMORY REGISTER OUTPUT

OCTAL REG #   DECIMAL REG #   HEXADECIMAL VALUE

        0            0             00000000
        1            1             00000000
        2            2             00000000
        3            3             00000000
        4            4             00000000
        5            5             00000000
        6            6             00000000
        7            7             00000000
       10            8             00000000
       11            9             00000000
       12           10             00000000
       13           11             00000000
       14           12             00000000
       15           13             00000000
       16           14             00000000
       17           15             00000000
       20           16             00000000
       21           17             00000000
       22           18             00000000
       23           19             00000000
```

```
 24         20              00000000
 25         21              00000000
 26         22              00000000
 27         23              00000000
 30      NOT USABLE OR ADDRESSABLE
 31      NOT USABLE OR ADDRESSABLE
 32      NOT USABLE OR ADDRESSABLE
 33      NOT USABLE OR ADDRESSABLE
 34      NOT USABLE OR ADDRESSABLE
 35      NOT USABLE OR ADDRESSABLE
 36      NOT USABLE OR ADDRESSABLE
 37      NOT USABLE OR ADDRESSABLE
 40      NOT USABLE OR ADDRESSABLE
 41      NOT USABLE OR ADDRESSABLE
 42      NOT USABLE OR ADDRESSABLE
 43      NOT USABLE OR ADDRESSABLE
 44      NOT USABLE OR ADDRESSABLE
 45      NOT USABLE OR ADDRESSABLE
 46      NOT USABLE OR ADDRESSABLE
 47      NOT USABLE OR ADDRESSABLE
 50      NOT USABLE OR ADDRESSABLE
 51      NOT USABLE OR ADDRESSABLE
 52      NOT USABLE OR ADDRESSABLE
 53      NOT USABLE OR ADDRESSABLE
 54      NOT USABLE OR ADDRESSABLE
 55      NOT USABLE OR ADDRESSABLE
 56      NOT USABLE OR ADDRESSABLE
 57      NOT USABLE OR ADDRESSABLE
 60         24              00000000
 61         24              00000000
 62         24              00000000
 63         24              00000000
 64         24              00000000
 65         24              00000000
 66         24              00000000
 67         24              00000000
 70         25              00053E07
 71         25              00053E07
 72         25              00053E07
 73         25              00053E07
 74         25              00053E07
 75         25              00053E07
 76         25              00053E07
 77         25              00053E07
100         26              F1F2F3F4
101         27              F5F6F7F8
102         28              00000000
103         29              00000008
```

```
104         30          00000008
105         31          00000004
106         32          00000001
107         33          00000000
110         34          FFFFFFFF
111         35          00000000
112         36          00000000
113         37          00000000
114         38          00000000
115         39          00000000
116         40          00000000
117         41          00000000
120         42          00000000
121         43          00000064
122         44          00000000
123         45          00000000
124         46          00000000
125         47          00000000
126         48          00000000
127         49          00000000
130      NOT USABLE OR ADDRESSABLE
131      NOT USABLE OR ADDRESSABLE
132      NOT USABLE OR ADDRESSABLE
133      NOT USABLE OR ADDRESSABLE
134      NOT USABLE OR ADDRESSABLE
135      NOT USABLE OR ADDRESSABLE
136      NOT USABLE OR ADDRESSABLE
137      NOT USABLE OR ADDRESSABLE
140         50          00000000
141         51          00000000
142         52          00000000
143         53          00010E06
144         54          0000007B
145         55          00000000
146         56          00000000
147         57          00020016
150         58          00000000
151         59          00000000
152         60          00000000
153         61          00000000
154         62          00000000
155         63          00000000
156         64          00000000
157         65          00000000
160         66          00000000
161         67          00000000
162         68          00000000
163         69          00000000
```

```
         164              70                  00000000
         165              71                  00000000
         166              72                  00000000
         167              73                  00000000
         170              74                  00000000
         171              75                  00000000
         172              76                  00000000
         173              77                  00000000
         174              78                  00000000
         175              79                  00000000
         176              80                  00000000
         177              81                  00000000
         200              82                  00020037
         201              83                  00017BC9

                  CPU NUMBER 1

            CONTROL MEMORY REGISTER OUTPUT

     OCTAL REG #  DECIMAL REG #  HEXADECIMAL VALUE

           0               0                  00000834
           1               1                  00000116
           2               2                  00000117
           3               3                  0000084D
           4               4                  FFFFFFFE
           5               5                  0000085D
           6               6                  00000000
           7               7                  00000000
          10               8                  00000000
          11               9                  0000085D
          12              10                  00000000
          13              11                  00000000
          14              12                  00000000
          15              13                  00000000
          16              14                  00000000
          17              15                  00000000
          20              16                  00000000
          21              17                  00000000
          22              18                  00000000
          23              19                  00000000
          24              20                  00000000
          25              21                  00000000
          26              22                  00000000
          27              23                  00000000
          30        NOT USABLE OR ADDRESSABLE
          31        NOT USABLE OR ADDRESSABLE
          32        NOT USABLE OR ADDRESSABLE
```

```
 33       NOT USABLE OR ADDRESSABLE
 34       NOT USABLE OR ADDRESSABLE
 35       NOT USABLE OR ADDRESSABLE
 36       NOT USABLE OR ADDRESSABLE
 37       NOT USABLE OR ADDRESSABLE
 40       NOT USABLE OR ADDRESSABLE
 41       NOT USABLE OR ADDRESSABLE
 42       NOT USABLE OR ADDRESSABLE
 43       NOT USABLE OR ADDRESSABLE
 44       NOT USABLE CR ADDRESSABLE
 45       NOT USABLE OR ADDRESSABLE
 46       NOT USABLE OR ADDRESSABLE
 47       NOT USABLE OR ADDRESSABLE
 50       NOT USABLE OR ADDRESSABLE
 51       NOT USABLE OR ADDRESSABLE
 52       NOT USABLE OR ADDRESSABLE
 53       NOT USABLE OR ADDRESSABLE
 54       NOT USABLE OR ADDRESSABLE
 55       NOT USABLE OR ADDRESSABLE
 56       NOT USABLE OR ADDRESSABLE
 57       NOT USABLE OR ADDRESSABLE
 60           24            00000000
 61           24            00000000
 62           24            00000000
 63           24            00000000
 64           24            00000000
 65           24            00000000
 66           24            00000000
 67           24            00000000
 70           25            00110E00
 71           25            00110E00
 72           25            00110E00
 73           25            00110E00
 74           25            00110E00
 75           25            00110E00
 76           25            00110E00
 77           25            00110E00
100           26            00000000
101           27            00000000
102           28            00000000
103           29            00000000
104           30            00000000
105           31            00000000
106           32            00000000
107           33            00000000
110           34            FFFFFFFF
111           35            00000000
112           36            00000000
```

```
113        37          00000000
114        38          00000000
115        39          00000000
116        40          00000000
117        41          00000000
120        42          00000000
121        43          00000000
122        44          00000000
123        45          00000000
124        46          00000000
125        47          00000000
126        48          00000000
127        49          00000000
130     NOT USABLE  OR  ADDRESSABLE
131     NOT USABLE  OR  ADDRESSABLE
132     NOT USABLE  OR  ADDRESSABLE
133     NOT USABLE  OR  ADDRESSABLE
134     NOT USABLE  OR  ADDRESSABLE
135     NOT USABLE  OR  ADDRESSABLE
136     NOT USABLE  OR  ADDRESSABLE
137     NOT USABLE  OR  ADDRESSABLE
140        50          00000000
141        51          00000000
142        52          00000000
143        53          00000000
144        54          00000000
145        55          00000000
146        56          00000000
147        57          00000000
150        58          00000000
151        59          00000000
152        60          00000000
153        61          00000000
154        62          00000005
155        63          00100006
156        64          00000000
157        65          00020438
160        66          FFFFFFFF
161        67          FFFFFFFF
162        68          00000000
163        69          00000000
164        70          00000000
165        71          00000000
166        72          00000000
167        73          00000000
170        74          00000000
171        75          00000000
172        76          00000000
```

```
     173          77              00000000
     174          78              00000000
     175          79              00000000
     176          80              00000000
     177          81              00000000
     200          82              00000006
     201          83              0001C7F3

**** END EXECUTION ****
```

```
BEGIN
EXTERNAL PROCEDURE INITPAGE (R14);  NULL;
EXTERNAL PROCEDURE GETPAGE  (R14);  NULL;
EXTERNAL PROCEDURE PUTPAGE  (R14);  NULL;
EXTERNAL PROCEDURE GETFRAME(R14);  NULL;
INTEGER STARTDUMP; FUNCTION BXLE(3,#8700); FUNCTION SETZONE(8,#96F0);
INTEGER ENDDUMP; FUNCTION ALR(1,#1E00); FUNCTION CLR(1,#1500);
LONG REAL SAVEX,ZERO=0;  FUNCTION SLR(1,#1F00);
ARRAY 255 INTEGER CPU0=(25(0),#10E00,8(0),_1,75(0),#110E00,8(0),_1,
                                       75(0),#210E00,8(0),_1,50(0));
INTEGER CPU1 SYN CPU0(340),CPU2 SYN CPU0(680);
INTEGER REGISTER INST SYN R9,  CPUBASE SYN R11,  OPCODE SYN R8,
  OPERAND SYN R10, ASR SYN R12;
ARRAY 3 INTEGER CPUCM = (@@CPU0,@@CPU1,@@CPU2);
ARRAY 12 INTEGER MDUMP = 12(_1);  ARRAY 6 INTEGER TRACEVAL=6(0);
INTEGER MEMLIMIT,MEMBASE,INDEXREGLOC,INDEXREGVAL,BASEREGLOC,IAW,IAWP,
  REPLACELOC,IAWADDR,IAWW,REPEATAVAL,REPLACESTOREVAL,REPEATINST,
  REPEATINDEX,REPEATCOUNT,REPEATSKIP=0,LOC=0,PREGVAL=0,IMONCLOCK=_1,
  REALCLOCK=0;
ARRAY 256 SHORT INTEGER PAGETABLE=256(0);
ARRAY 3 SHORT INTEGER CPUSTATE = 3(1);
SHORT INTEGER ANEXT,ABNEXT,ACCG,BASEG,INDEXG,PPAGES=0,VPAGES=0;
SHORT INTEGER MEMCASE=1;
ARRAY 3 SHORT INTEGER INTLOCKOUT;
ARRAY 131 BYTE CMRADDR=(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,
  19,20,21,22,23,24(255),8(24),8(25),26,27,28,29,30,31,32,33,34,35,36,
  37,38,39,40,41,42,43,44,45,46,47,48,49,8(255),50,51,52,53,54,55,56,57,
  58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,
  81,82,83,84);
ARRAY 64 BYTE REPEATOP=(6,1,4,5,6,3,3,3,1,3,1,1,1,1,1,1,1,1,1,1,1,3,1,
  1,6,6,5,5,5,5,5,5,1,1,2,2,2,2,2,2,3,3,3,3,1,1,1,1,3,3,3,3,3,3,3,3,3,3,
  3,3,3,3,3,3);
ARRAY 8 BYTE REPEATOP02=(1,5,3,3,1,1,1,3);
ARRAY 16 BYTE HEX = (240,241,242,243,244,245,246,247,248,249,193,194,
  195,196,197,198);
INTEGER B7ADDR;
ARRAY 132 BYTE WBUF=132(" "); ARRAY 132 BYTE BLANK= 132(" ");
ARRAY 80 BYTE CBUF;
BYTE NCPU=#01,PAGING=#00,IDESIGN=#00,HWFLAG=#00;
BYTE REPEAT=#00,REPLACE=#00,COMPARE=#00,CHARADDR=#00,EXECSTATE=#00,
  INDEX=#00,REPEATCMR=#00;
BYTE ECHOL=#FF,ECHOI=#FF,FORMAT1=#FF,TRACEDO=#00,TRACE=#00,IDIR=#00;
```

```
PROCEDURE  BRANCH (R14);
BEGIN
  CASE R1 OF
  BEGIN
    GOTO EXECREMOTE;
    GOTO SETUP;
    GOTO WIPEOUT;
  END;
END;



PROCEDURE COMPUTESPECY (R14);
BEGIN  COMMENT THIS PROCEDURE COMPUTES THE OPERAND ADDRESS ACCORDING TO
  THE OPTIONAL INDIRECT ADDRESS WORD FORMAT;
  ARRAY 2 INTEGER REGSAVYCOM;  INTEGER SAV14;
  SAV14 := R14;  STM(R3,R4,REGSAVYCOM);
  OPERAND := INST AND #FFFF;  COMMENT SY ADDRESS;
  R4 := INST AND #E0000 SHRL 15;  COMMENT B VALUE IN INSTRUCTION;
  IF REPLACE THEN
  BEGIN
    R3 := CPUBASE + BASEG + 24;  REPLACELOC := R3;
    R3 := B3 AND #3FFFF;  R3 := R3 + OPERAND;
    REPLACESTOREVAL := R3;
  END;
  R3 := R4 + BASEG + CPUBASE;  BASEREGLOC := R3;
  OPERAND := OPERAND + B3 AND #3FFFF;  COMMENT SY + (S(B));
  R3 := IAW SHLL 2;
  SET(INDEX);
  IF R3 < 0 THEN
  BEGIN  COMMENT GET B(B);
    IF R4 = 0 THEN GOTO NOINDEXREG;
    R4 := R4 + CPUBASE + INDEXG;  INDEXREGLOC := R4;
    R4 := B4 AND #FFFF;  INDEXREGVAL := R4;
    OPERAND := OPERAND + R4 AND #3FFFF;  GOTO FINISH;
  END;
NOINDEXREG:R4 := R4 - R4;  INDEXREGVAL := R4;  RESET(INDEX);
FINISH:LM(R3,R4,REGSAVYCCM);  R14 := SAV14;
END;
```

```
PROCEDURE COMPUTEY (R14);
BEGIN  COMMENT THIS PROCEDURE COMPUTES OPERAND ADDRESS Y BY ADDING Y +
  B(B) + S(S) OF IVSTRUCTION WORD;
  INTEGER SAV14;
  ARRAY 2 INTEGER REGSAVCOM;
  STM(R3,R4,REGSAVCOM);  SAV14 := R14;
  OPERAND := INST AND #1FFF;   COMMENT Y VALUE;
  R3 := INST SHLL 12 SHRL 29;   COMMENT INDEX REGISTER NUMBER;
  SET(INDEX);
  IF R3 = 0 THEN
  BEGIN  INDEXREGVAL := R3;  RESET(INDEX);  GOTO NOINDEXREGS;  END;
  R4 := R3 SHLL 2 + CPUBASE + INDEXG;  COMMENT ADDR OF INDEX REGISTER;
  R3 := B4 AND #FFFF;
  INDEXREGVAL := R3;  INDEXREGLOC := R4;
  OPERAND := OPERAND + R3;    COMMENT Y + INDEX REGISTER VALUE;
NOINDEXREGS:R3 := INST SHLL 16 SHRL 29;  COMMENT BASE REGISTER NUMBER;
  IF REPLACE THEN  COMMENT OPERAND ADDRESS USES S(6) VICE S(S) FOR
  REPLACE INSTRUCTIONS IN THE REPEAT MODE;
  BEGIN
    R4 := CPUBASE + BASEG + 24;  REPLACELOC := R4;
    R4 := B4 AND #3FFFF;
    R4 := R4 + OPERAND;  REPLACESTOREVAL := R4;
  END;
  R4 := R3 SHLL 2 + CPUBASE + BASEG;  COMMENT ADDRESS OF BASE REGISTER;
  BASEREGLOC := R4;
  R3 := B4 AND #3FFFF;  CCMMENT GET THE VALUE;
  OPERAND := OPERAND + R3 AND #3FFFF;
  LM(R3,R4,REGSAVCOM);  R14 := SAV14;
END;


PROCEDURE ADD (R14);  COMMENT ADDS R2 AND R3 AND RETURNS VALUE IN R6
                      ALSO SETS FIXED POINT OVERFLOW IF NEEDED;
BEGIN
  ASR := ASR AND #FFFFFFF7;  R6 := R2;  ALR(R6,R3);
  IF OVERFLOW OR > THEN R6 := R6 + 1;
  IF R2 > 0 AND R3 > 0 AND R6 < 0 THEN ASR := ASR OR #08;
  IF R2 < 0 AND R3 < 0 AND R6 > 0 THEN ASR := ASR OR #08;
END;
```

```
PROCEDURE NORMREAD (R14);
BEGIN   COMMENT USED FOR OPCODES 10 - 47;
ARRAY 3 INTEGER SAVENREAD;  INTEGER SAVE14;
  STM(R2,R4,SAVENREAD);  SAVE14 := R14;
  R6 := R6 + 1;
  CASE R6 OF
  BEGIN
    BEGIN
      R4 := ASR;  R2 := INDEXREGVAL AND #FFFF;          COMMENT K = 0;
      R3 := INST SHLL 16 SHRA 16;  ADD;
      ASR := R4;  R6 := OPERAND;
    END;
    OPERAND := OPERAND SHLL 16 SHRA 16;                 COMMENT K = 1;
    OPERAND := OPERAND SHRA 16;                         COMMENT K = 2;
    NULL;                                               COMMENT K = 3;
    OPERAND := OPERAND AND #FF;                         COMMENT K = 4;
    OPERAND := OPERAND SHRL 8 AND #FF;                  COMMENT K = 5;
    OPERAND := OPERAND SHRL 16 AND #FF;                 COMMENT K = 6;
    OPERAND := OPERAND SHRL 24;                         COMMENT K = 7;
  END;
  LM(R2,R4,SAVENREAD);  R14 := SAVE14;
END;


PROCEDURE WRITSTRING (R14);    COMMENT THIS PROCEDURE WRITES A STRING
      PREVIOUSLY PUT IN THE WBUF & RESTORES THE REGS & BLANKS THE WBUF;
   BEGIN
  ARRAY 4 INTEGER REGWRIT;
  STM(R14,R1,REGWRIT);
      R0 := @WBUF;  WRITE;  MVC(131,WBUF,BLANK);
  LM(R14,R1,REGWRIT);
   END;        COMMENT END OF WRITSTRING;
```

```
PROCEDURE INTERRUPT (R14);
  BEGIN
  MEMCASE := R1;
  CASE R1 OF
  BEGIN      COMMENT START OF CASE STATEMENT;
    BEGIN                                    COMMENT CLASS 1 INTERRUPT;
      MEM(R11 + 204) := ASR;                 COMMENT STORE ASR IN DSW;
      ASR := ASR OR #87E80 AND #7FFF80; COMMENT MODIFY THE ASR,NDRO ON;
      MEM(R11 + 100) := ASR;                 COMMENT STORE ASR IN THE ASR;
      MEM(R11 + 208) := R2;                  COMMENT STORE ISC IN DSW;
      R3 := MEM(R11 + 328);                  COMMENT LOAD FROM THE P REG;
      MEM(R11 + 212) := R3;                  COMMENT STORE P IN DSW;
      R3 := MEM(R11 + 200);                  COMMENT LOAD BRANCH ADDR;
      MEM(R11 + 328) := R3;                  COMMENT LOAD P REG;
    END;                                     COMMENT END CLASS 1 INTERRUPT;
    BEGIN                             COMMENT CLASS 2 INTERRUPT;
      R3 := ASR SHLL 13;
      IF R3 < 0 THEN GOTO ENDINTER;          COMMENT CLASS 2 INHIBITED;
      MEM(R11 + 212) := ASR;                 COMMENT STORE ASR IN DSW;
      ASR := ASR OR #43E00 AND #7FFF80;              COMMENT MODIFY ASR;
      MEM(R11 + 100) := ASR;                 COMMENT STORE ASR IN THE ASR;
      MEM(R11 + 224) := R2;                  COMMENT STORE ISC IN DSW;
      R3 := MEM(R11 + 328);                  COMMENT LOAD FROM P REG;
      IF R2 = 2 THEN  R3 := R3 - 1;  COMMENT P - 1 FOR ILL. INST;
      MEM(R11 + 228) := R3;                  COMMENT STORE P IN DSW;
      R3 := MEM(R11 + 216);                  COMMENT LOAD BRANCH ADDRESS;
      MEM(R11 + 328) := R3;                  COMMENT LOAD P REG;
    END;                                     COMMENT  END CLASS 2 INTERRUPT;
    BEGIN                                    COMMENT CLASS 3 INTERRUPT;
      R3 := ASR SHLL 14;
      IF R3 < 0 THEN GOTO ENDINTER;          COMMENT CLASS 3 INHIBITED;
      MEM(R11 + 236) := ASR;                 COMMENT STORE ASR IN DSW;
      ASR := ASR OR #21E00 AND #7FFF80;              COMMENT MODIFY ASR;
      MEM(R11 + 100) := ASR;                 COMMENT STORE ASR IN THE ASR;
      MEM(R11 + 240) := R2;                  COMMENT STORE ISC IN DSW;
      R3 := MEM(R11 + 328);                  COMMENT LOAD FROM P REG;
      MEM(R11 + 244) := R3;                  COMMENT STORE P IN DSW;
      R3 := MEM(R11 + 232);                  COMMENT LOAD BRANCH ADDRESS;
      MEM(R11 + 328) := R3;                  COMMENT LOAD THE P REG;
    END;                                     COMMENT END CLASS 3 INTERRUPT;
    BEGIN                                    COMMENT CLASS 4 INTERRUPT;
      MEM(R11 + 252) := ASR;                 COMMENT STORE ASR IN DSW;
      ASR := ASR OR #10E00 AND #7FFF80;             COMMENT MODIFY ASR;
      MEM(R11 + 100) := ASR;                 COMMENT STORE ASR IN THE ASR;
      MEM(R11 + 256) := R2;                  COMMENT STORE ISC IN DSW;
      R3 := MEM(R11 + 328);                  COMMENT LOAD FROM P REG;
      MEM(R11 + 260) := R3;                  COMMENT STORE P IN DSW;
      R3 := MEM(R11 + 248);                  COMMENT LOAD BRANCH ADDRESS;
```

```
      MEM(R11 + 328) := R3;                  COMMENT LOAD THE P REG;
     END;                                    COMMENT END CLASS 4 INTERRUPT;
   END;    COMMENT END OF CASE STATEMENT;
ENDINTER:R1 := 2;  BRANCH;
   END;                                      COMMENT END INTERRUPT;
```

```
SEGMENT PROCEDURE MEMORY (R14);
  COMMENT SIMULATES AN/UYK-7 MEMORY.  THIS PROCEDURE ACCOMPLISHES
  ALL MEMORY ACCESSES AND DOES PAGING OF THE SIMULATED AN/UYK-7
  MEMORY AS REQUIRED BY THE CPU AND IOC;
BEGIN ARRAY 9 INTEGER REGSAVEM;  INTEGER SAVE14;


PROCEDURE FETCHPAGE (R14);
  COMMENT LOADS A PAGE * 2 SPECIFIED BY R2 INTO MEMORY;
BEGIN INTEGER SAVE14;  ARRAY 9 INTEGER SAVEREGS;
  SAVE14 := R14;  STM(R0,R8,SAVEREGS);
  R14 := @PAGETABLE;
  R4 := 2;  R5 := VPAGES - 1 SHLL 1 + R14;
       COMMENT R4 = INCREMENT, R5 = COMPARAND;
  FOR R1 := 224 STEP _32 UNTIL 128 DO  COMMENT R1 = MASK FOR CLI INST;
  BEGIN
    R3 := @PAGETABLE;
    BALR(R14,R0);
    EX(R1,CLI(0,B3));
    IF = THEN GOTO FOUND;
    BXLE(R3,R4,B14);
  END;
FOUND:  LH(R4,B3);  R0 := R4;  R4:= R4 AND #4000;
  R0 := R0 AND #FF SHLL 12 + MEMBASE;
  R6 := @PAGETABLE;
    R1 := R3 - R6 SHRL 1;              COMMENT R1 = PAGE # TO GO;
  IF R4 = 0 THEN      COMMENT DIRTY PAGE, MUST WRITE OUT;
  BEGIN
    PUTPAGE;
  END;
  R4 := 2;  COMMENT SET ALL PAGES UNREFERENCED;
  BALR(R14,R0);
  OI(32,B6);
  BXLE(R6,R4,B14);
  LH(R7,B3);          COMMENT SET OLD PAGE NOT IN CORE;
  R6 := #8000 XOR R7;
  STH(R6,B3);  R1 := R1 SHLL 1;
  R6 := PAGETABLE(R1) AND #FF OR #E000;
  PAGETABLE(R2) := R6;
  R1 := R2 SHRL 1;  COMMENT ACTUAL PAGE #;
  GETPAGE;
  R14 := SAVE14;
  R14 := SAVE14;  LM(R0,R8,SAVEREGS);
END;


PROCEDURE MEMINTERRUPT (R14);
  COMMENT SET UP INTERRUPT CALLS FOR MEMORY ACCESSES OUTSIDE OF RANGE;
```

```
BEGIN  INTEGER SAVE14;  ARRAY 9 INTEGER SAVEREGS;
  SAVE14 := R14;  STM(R0,R8,SAVEREGS);
  R5 := R5 SHRL 14 SHLL 4;  R1 := MEMCASE;
  CASE R1 OF
  BEGIN
    R2 := R5 OR 2;          COMMENT INSTRUCTION FETCH CASE;
    R2 := R5;               COMMENT OPERAND FETCH CASE;
    R2 := R5;               COMMENT OPERAND STORE CASE;
    R2 := R5 OR 10;         COMMENT IOC MEMORY FETCH;
    R2 := R5 OR 10;         COMMENT IOC MEMORY STORE;
    R2 := R5;               COMMENT UNCONDITIONAL NORMSTORE MEMORY FETCH;
    R2 :=.R5;              COMMENT INDIRECT ADDR FETCH;
  END;
  R1 := 1;  INTERRUPT;
  R14 := SAVE14;  LM(R0,R8,SAVEREGS);
END;


PROCEDURE MEMFETCH (R14);
  COMMENT FETCH THE AN/UYK-7 MEMORY ADDRESS IN R5, RETURN VALUE IN R8;
BEGIN INTEGER SAVE14;  ARRAY 8 INTEGER SAVEREGS;
  SAVE14 := R14;  STM(R0,R7,SAVEREGS);
  R14 := MEMLIMIT;
  IF R5 >= R14 THEN MEMINTERRUPT;
  IF ¬ PAGING THEN
  BEGIN     COMMENT BYPASS PAGING AND COMPUTE 360 ADDRESS;
    R8 := R5 SHLL 2 + MEMBASE;  R8 := B8;  GOTO ENDFETCH;
  END;
  R2 := R5 SHRL 10 SHLL 1;  COMMENT R2 = PAGE # SHLL 1;
  R5 := R5 AND #3FF SHLL 2;     COMMENT R5 = LINE #;
SETR3: R3 := PAGETABLE(R2);      COMMENT R3 = TABLE ENTRY FOR PAGE(R2);
  IF R3 < 0 THEN
  BEGIN
    R4 := R3 AND #C0FF;
    PAGETABLE(R2) := R4;       COMMENT MAKE IT A REFERENCED PAGE;
    R3 := R3 AND #FF SHLL 12 + MEMBASE + R5;   COMMENT R3 = 360 ADDR;
    R8 := B3;  GOTO ENDFETCH;
  END;
  FETCHPAGE;  GOTO SETR3;
ENDFETCH: R14 := SAVE14;  LM(R0,R7,SAVEREGS);
END;


PROCEDURE MEMSTORE (R14);
  COMMENT STORE THE VALUE PASSED IN R2 INTO ADDRESS PASSED IN OPERAND;
BEGIN INTEGER SAVE14;  ARRAY 9 INTEGER SAVEREGS;
  SAVE14 := R14;  STM(R0,R8,SAVEREGS);
  R14 := MEMLIMIT;
```

```
  IF OPERAND >= R14 THEN BEGIN  R5 := OPERAND;  MEMINTERRUPT;  END;
  IF ¬ PAGING THEN      COMMENT NO PAGING, BYPASS PAGING;
  BEGIN
   R8 := OPERAND SHLL 2 + MEMBASE;  B8 := R2;  GOTO ENDSTORE;
  END;
  R3 := OPERAND SHRL 10 SHLL 1;  COMMENT R3=VIRTUAL PAGE # SHLL1;
  R4 := OPERAND AND #3FF SHLL 2:      COMMENT R4 = LINE #;
SETR5: R5 := PAGETABLE(R3);
  IF R5 < 0 THEN             COMMENT PAGE IS IN MEMORY;
  BEGIN
   R6 := R5 AND #80FF;  PAGETABLE(R3) := R6;  COMMENT DIRTY PAGE;
   R6 := R5 AND #FF SHLL 12 + MEMBASE + R4;  COMMENT 360 ADDRESS;
   B6 := R2;  GOTO ENDSTORE;
  END;
  R6 := R2;  R2 := R3;
  FETCHPAGE;  R2 := R6;  GOTO SETR5;
ENDSTORE: R14 := SAVE14;  LM(RO,R8,SAVEREGS);
END;


  SAVE14 := R14;  STM(RO,R8,REGSAVEM);
  R1 := R1 + 1;  MEMCASE := R1;
CASESTART:CASE R1 OF
  BEGIN
   BEGIN    COMMENT INSTRUCTION FETCH;
     R7 := MEM(CPJBASE + 328);    COMMENT P REG IN R7;
     R2 := R7 AND #FFFF;   COMMENT R2 = DISPLACEMENT;
     R4 := R7 AND #E0000 SHRL 17 SHLL 2; COMMENT B REG WITH DISPLACE;
     R3 := ASR SHLL 20;  COMMENT BIT 11 OF ASR IN SIGN BIT OF R3;
     IF R3 < 0 THEN
     BEGIN
       R3 := CPUBASE + R4 + 168;  GOTO NOPROCK;
     END;
     R3 := CPUBASE + R4 + 64;
     IF EXECSTATE THEN GOTO NOPROCK;
     R5 := ASR AND #200;  IF R5 > 0 THEN GOTO NOPROCK;
     R5 := R3 + 200;  COMMENT R5 = ADDRESS OF SPR;
     R5 := B5 SHLL 11;  COMMENT R5 = SPR CONTENTS WITH SHIFT;
     IF R5 >= 0 THEN
     BEGIN  R1 := 2;  R2 := 13;  INTERRUPT;  END;
     R5 := R5 SHRL 11 AND #FFFF;
     IF R2 > R5 THEN
     BEGIN  R1 := 2;  R2 := 14;  INTERRUPT;  END;
NOPROCK:  R5 := B3 + R2;     COMMENT R5 = UYK-7 ADDR;
     R2 := MEM(CPUBASE + 96) SHLL 13;  COMMENT R2 HAS BREAKPOINT;
     IF R2 < 0 THEN
     BEGIN            COMMENT CHECK BREAKPOINT ADDRESS;
       R2 := R2 SHLL 1 SHRL 14;  COMMENT R2 COTAINS BREAKPOINT ADDR;
```

```
        IF R2 = R5 THEN
        BEGIN
          R6 := MEM(CPUBASE +336) AND #F0000;  COMMENT LOAD SWITCHES;
          IF R6 > 0 THEN
          BEGIN
            R6 := ASR SHRL 20 SHLL 1;  R1 := 1;  CPUSTATE(R6) := R1;
            MEM(CPUBASE + 100) := ASR;  R1 := 2;  BRANCH;
          END;
          R1 := 2;  R2 := 11;  INTERRUPT;
        END;
      END;
      IF TRACEDO THEN
      BEGIN
        R1 := ASR SHRL 20 AND #07 + 1;
        CASE R1 OF
        BEGIN
          BEGIN  R2 := R2 - R2;  R3 := 4;  END;
          BEGIN  R2 := 8;  R3 := 12;  END;
          BEGIN  R2 := 16;  R3 := 20;  END;
        END;  R2 := TRACEVAL(R2);  R3 := TRACEVAL(R3);
        IF R2 <= R5 AND R3 >= R5 THEN
        BEGIN
          SET(TRACE);  PREGVAL := R7;  LOC := R5;
        END ELSE
          RESET(TRACE);
      END;
      MEMFETCH;  R7 := R7 + 1 AND #EFFFF;
      MEM(CPUBASE + 328) := R7;  INST := R8;
    END;
    BEGIN            COMMENT OPERAND FETCH CASE;
      R7 := BASEREGLOC;  R7 := B7;  COMMENT R7 = BASE REG CONTENTS;
      R6 := ASR AND #A00;
      IF R6 > 0 OR EXECSTATE THEN GOTO NOCHK;
      R5 := BASEREGLOC + 200;  R5 := B5;  COMMENT R5 = SPR;
      R6 := R5 SHLL 12;
      IF R6 >= 0 THEN
      BEGIN
        R1 := 2;  R2 := 6;  INTERRUPT;
      END;
      R6 := R5 AND #FFFF + R7;  COMMENT R6 = MAX ADDR OF SEGMENT;
      IF OPERAND > R6 THEN
      BEGIN
        R1 := 2;  R2 := 10;  INTERRUPT;
      END;
NOCHK: R6 := MEM(CPUBASE + 96) SHLL 12;  COMMENT START BREAKPOINT CHECK;
      IF R6 < 0 THEN
      BEGIN
        R6 := R6 SHRL 12 AND #3FFFF;
```

```
          IF OPERAND = R6 THEN
          BEGIN
            R4 := MEM(CPUBASE + 336) AND #F0000;  COMMENT CHECK SWITCH;
            IF R4 > 0 THEN
            BEGIN
              R6 := ASR SHRL 20 SHLL 1;   R1 := 1;  CPUSTATE(R6) := R1;
              MEM(CPUBASE + 100) := ASR;  R1 := 2;  BRANCH;
            END;
            R1 := 2;  R2 := 5;  INTERRUPT;
          END;
        END;
        R5 := OPERAND;  MEMFETCH;  OPERAND := R8;
        IF CHARACDR THEN
        BEGIN    COMMENT CHARACTER ADDRESSING;
          R5 := IAWP;  R4 := 32 - IAWW;
          OPERAND := OPERAND SHRL R5 SHLL R4 SHRL R4;
        END;
      END;
      BEGIN      COMMENT OPERAND STORE CASE;
        R7 := BASEREGLOC;  R7 := B7;
        R6 := ASR AND #A00;
        IF R6 > 0 OR EXECSTATE THEN GOTO NOPROTEST;
        R5 := BASEREGLOC + 200;  R5 := B5;  COMMENT R5 = SPR;
        R6 := R5 SHLL 13;
        IF R6 >= 0 THEN   COMMENT OPERAND WRITE INTERRUPT;
        BEGIN
          R1 := 2;  R2 := 9;  INTERRUPT;
        END;
        R6 := R5 AND #FFFF + R7;    COMMENT R6 = MAX ADDRESS;
        IF OPERAND > R6 THEN
        BEGIN         COMMENT OPERAND LIMIT INTERRUPT;
          R1 := 2;  R2 := 10;  INTERRUPT;
        END;
NOPROTEST: R6 := MEM(CPUBASE +  96) SHLL 12;  COMMENT BREAKPOINT;
        IF R6 < 0 THEN
        BEGIN
          R6 := R6 SHRL 12 AND #3FFFF;
          IF OPERAND = R6 THEN
          BEGIN
            R4 := MEM(CPUBASE + 336) AND #F0000;  COMMENT CHECK SWITCHES;
            IF R4 > 0 THEN
            BEGIN
              R6 := ASR SHRL 20 SHLL 1;   R1 := 1;  CPUSTATE(R6) := R1;
              MEM(CPJBASE + 100) := ASR;  R1 := 2;  BRANCH;
            END;
            R1 := 2;  R2 := 5;  INTERRUPT;
          END;
        END;
```

```
      IF CHARADDR THEN
      BEGIN
        R5 := OPERAND;  MEMFETCH;  COMMENT NEED OLD OPERAND (IN R8);
        R6 := 32 - IAWW;
        R5 := R6 - IAWP;
        R4 := _1 SHLL R6 SHRL R5 XOR _1;  COMMENT BUILD MASK;
        R2 := R2 SHLL R6 SHRL R5;         COMMENT BUILD CHARACTER;
        R5 := R8 AND R4 OR R2;         COMMENT BUILD FINAL WORD;
        R2 := R5;
      END;
      MEMSTORE;
    END;
    BEGIN      COMMENT IOC FETCH FROM MEMORY;
      R5 := R4;  MEMFETCH;  REGSAVEM(8) := R8;
    END;
    BEGIN       COMMENT IOC STORE INTO MEMORY;
      R3 := OPERAND;  OPERAND := R4;  MEMSTORE;  OPERAND := R3;
    END;
    BEGIN        COMMENT UNCONDITIONAL FETCH FOR NORMSTORE;
      R5 := OPERAND;  MEMFETCH;  OPERAND := R8;
    END;
    BEGIN      COMMENT INDIRECT ADDRESSING, OPERAND FETCH;
      IF EXECSTATE THEN GOTO ENDIACHK;
      R1 := ASR SHRL 20;
      IF R1 < 0 THEN
      BEGIN
        R2 := BASEREGLOC + 96;  R2 := B2 SHLL 14;
        IF R2 >= 0 THEN
        BEGIN  R1 := 2;   R2 := 6;    INTERRUPT;    END;
        R2 := R2 SHLL 1;
        IF R2 >= 0 THEN
        BEGIN  R1 := 2;   R2 := 6;    INTERRUPT;    END;
      END ELSE
      BEGIN
        R2 := BASEREGLOC + 200;  R2 := B2 SHLL 14;
        IF R2 >= 0 THEN
        BEGIN
          R1 := 2;  R2 := 6;  INTERRUPT;
        END;
      END;
ENDIACHK: R1 := 2;  GOTO CASESTART;  COMMENT GO CHECK OPERAND FETCH;
    END;
  END;
  LM(R0,R8,REGSAVEM);  R14 := SAVE14;
END;   COMMENT END MEMORY PROCEDURE;
```

```
PROCEDURE NORMSTORE (R14);
BEGIN
  INTEGER SAVE14;  SAVE14 := R14;
  R4 := OPERAND;  R1 := 5;  MEMORY;
  R6 := R6 + 1;
  CASE R6 OF
  BEGIN
    GOTO FINISH;                                            COMMENT K = 0;
    BEGIN                                                   COMMENT K = 1;
      R5 := R5 AND #FFFF;  OPERAND := OPERAND AND #FFFF0000 OR R5;
    END;
    BEGIN                                                   COMMENT K = 2;
      R5 := R5 SHLL 16;  OPERAND := OPERAND AND #FFFF OR R5;
    END;
    OPERAND := R5;                                          COMMENT K = 3;
    BEGIN                                                   COMMENT K = 4;
      R5 := R5 AND #FF;  OPERAND := OPERAND AND #FFFFFF00 OR R5;
    END;
    BEGIN                                                   COMMENT K = 5;
      R5 := R5 AND #FF SHLL 8;  OPERAND := OPERAND AND #FFFF00FF OR R5;
    END;
    BEGIN                                                   COMMENT K = 6;
      R5 := R5 AND #FF SHLL 16;  OPERAND := OPERAND AND #FF00FFFF OR R5;
    END;
    BEGIN                                                   COMMENT K = 7;
      R5 := R5 AND #FF SHLL 24;  OPERAND := OPERAND AND #FFFFFF OR R5;
    END;
  END;
  R2 := OPERAND;  OPERAND := R4;  R1 := 2;  MEMORY;
FINISH:  R14 := SAVE14;
END;


PROCEDURE TIME (R14);
BEGIN  ARRAY 4 INTEGER TSAVE;  STM(R11,R14,TSAVE);
  R12 := MEM(CPUBASE + 332) + R8;  COMMENT SIMTIME + INSTRUCTION TIME;
  MEM(CPUBASE + 332) := R12;
  R12 := MEM(CPUBASE + 136);
  IF R12 < 0 THEN GOTO ENDT;
  R12 := R12 - R8;  MEM(CPUBASE + 136) := R12;
  IF R12 < 0 THEN
  BEGIN
    R1 := 2;  R2 := 15;  INTERRUPT;
  END;
ENDT:  LM(R11,R14,TSAVE);
END;
```

```
PROCEDURE DUMPREGS (R14);
BEGIN
  COMMENT THIS PROCEDURE DUMPS THE CMR REGISTERS;
  INTEGER SAVE14; ARRAY 8 INTEGER REGSAVED;
  SAVE14 :=  R14; STM(R0,R8,REGSAVED);
  R0 := @WBUF;  WRITE;
  MVC(29,WBUF(6),"CONTROL MEMORY REGISTER OUTPUT");
  R0 := @WBUF;  WRITE;  MVC(60,WBUF,BLANK);  WRITE;
  MVC(44,WBUF,"OCTAL REG #  DECIMAL REG #  HEXADECIMAL VALUE");
  R0 := @WBUF;  WRITE;  MVC(44,WBUF,BLANK);  WRITE;
  R1 := STARTDUMP AND #1FF;  R2 := ENDDUMP AND #1FF;  R7 := #F0;
  R8 :=@SAVEX;
  WHILE R1 <= R2 DO
  BEGIN
    R3 :=R1 SHRL 6 OR R7;
    IF R3 = R7 THEN R3 := #40; STC(R3,WBUF(4));
    R4 := R1 AND #38 SHRL 3 OR R7;
    IF R3 = #40 AND R4 = R7 THEN R4 := #40; STC(R4,WBUF(5));
    R5 := R1 AND #07 OR R7;  STC(R5,WBUF(6));
    R3 := R3 -R3;  IC(R3,CMRADDR(R1));
    IF R3 = 255 THEN
    BEGIN
      MVC(24,WBUF(13),"NOT USABLE OR ADDRESSABLE");
      GOTO OUTPUT;
    END;
    CVD(R3,SAVEX);  R4 := R4 - R4;  R5 := R4;  R6 := #40;
    IC(R4,MEM(R8 + 6));  IC(R5,MEM(R8 + 7));  R4 := R4 OR R7;
    IF R4 = R7 THEN STC(R6,WBUF(17)) ELSE STC(R4,WBUF(17));
    R5 := R5 SHRL 4  OR R7;  STC(R5,WBUF(18));
    R6 := 39;  R5 := R3 SHLL 2 + CPUBASE;  R3 := B5;
    FOR R5 := 0 STEP 4 UNTIL 28 DO
    BEGIN
      R4 := R3 SHRL R5 AND #0F;  IC(R4,HEX(R4));
      STC(R4,WBUF(R6));  R6 := R6 - 1;
    END;
OUTPUT:R0 := @WBUF;  WRITE;  MVC(50,WBUF,BLANK);  R1 := R1 + 1;
  END;
  LM(R0,R8,REGSAVED);  R14 := SAVE14;
END;
```

```
SEGMENT PROCEDURE IOC (R14);           COMMENT OPERAND = R10, INST = R9,
                                   A * 4 IN R7, CPUBASE = R11,ASR = R12;
         COMMENT  MEMORY CASES:    R1 = 3 FETCH MEM(R4) --> R2,
                                   R1 = 4 STORE R2 --> MEM(R4);
BEGIN
  ARRAY 1 INTEGER ICBUF SYN CBUF;
  ARRAY 1 INTEGER IWBUF SYN WBUF;
  ARRAY 16 INTEGER SAVIO;
  ARRAY 128 INTEGER IOCMEM;
  BYTE CHAIN, MONITOR, CAR;
  STM(R0,R15,SAVIO);
GET:R4 := OPERAND;
  R1 := 3;  MEMORY;  INST := R2;          COMMENT FETCH IOC INST IN INST;
GOT:  OPCODE := INST SHRL 26 - 7;         COMMENT SCALE FOR CASE STM.;
  IF OPCODE < 1 OR OPCODE > 16 THEN
  BEGIN                                  COMMENT ILLEGAL OPCODES;
    OPCODE := 10;  GOTO STARTIO;
  END;
  R6 := INST SHLL 6 SHRL 30;              COMMENT K DESIG IN R6;
  R5 := INST SHLL 8 SHRL 28;              COMMENT J FIELD IN R5;
  R4 := INST AND #3FFFF;                  COMMENT OPERAND ADDR;
  INST := INST SHLL 12;
  IF INST < 0 THEN SET(MONITOR)
    ELSE RESET(MONITOR);
  INST := INST SHLL 1;
  IF INST < 0 THEN SET(CHAIN)  ELSE RESET(CHAIN);
  R1 := 3;  MEMORY;  R3 := R2;            COMMENT R3 NOW HAS OPERAND;
  R2 := R5 SHLL 3;
  IF OPCODE = 1 AND R5 ¬= 5 THEN
BEGIN  OPCODE := 10;  RESET(CAR);  END;
STARTIO:
  CASE OPCODE OF
  BEGIN
    BEGIN                                 COMMENT OPCODE 10 (INPUT);
      RESET(CAR);
      R4 := OPERAND + 1;                  COMMENT UPDATE ADDR;
      R1 := R4 SHLL 6;                    COMMENT BUILD CAP;
      R6 := R6 SHLL 4;                   COMMENT SHIFT K TO PROPER FIELD;
      IF CHAIN THEN R1 := R1 OR #08;
      IF MONITOR THEN R1 := R1 OR #04;
      R1 := R1 OR R6;
      IOCMEM(R2) := R1;                 COMMENT STORE CAP,K,C,M IN IOCMEM;
      IOCMEM(R2 + 4) := R3;             COMMENT STORE BFA & BCA IN IOCMEM;
      GOTO BUFFER1;
    END;
    BEGIN                                 COMMENT OPCODE 11 (OUTPUT);
      IF R5 ¬= 6 THEN                    COMMENT ILLEGAL INSTRUCTION;
      BEGIN  OPCODE := 10;  RESET(CAR);  GOTO STARTIO;  END;
```

```
    R2 := R2 + 128;                    COMMENT IOCMEM INDEX FOR OUTPUT;
    OPCODE := 1;                       COMMENT SET OPCODE;
    GOTO STARTIO;
  END;
  NULL;                               COMMENT NO EXTERNAL FUNCTION 12;
  NULL;                               COMMENT NO EXTERNAL INTERRUPT 13;
  NULL;                               COMMENT NO TERMINATE OPCODE 14;
  BEGIN                               COMMENT INTERRUPT OPCODE 15;
    R14 := ASR SHRL 20 SHLL 1;        COMMENT GET CP NO.;
    R1 := 31 - R5;                    COMMENT SHIFT FOR LOCKOUT REG.;
    R2 := INTLOCKOUT(R14) SHLL R1;    COMMENT CHECK FOR ENABLE ON CH.J;
    IF R2 < 0 THEN
    BEGIN
      R1 := 3;  R2 := #0B;  INTERRUPT;
    END;
  END;
  BEGIN                               COMMENT ACTIVATE CHAIN  16;
    RESET(CAR);
    IF R6 = 1 AND R5 ¬= 6 THEN GOTO ILLEGALINST;
    IF R6 =0 AND R5 ¬= 5 THEN GOTO ILLEGALINST;
    R2 := R4 SHLL 6;                  COMMENT BUILD CAP FROM Y;
    R1 := R6 SHLL 4 + R5 SHLL 3;      COMMENT INDEX INTO IOCMEM;
    R3 := IOCMEM(R1) AND #3F;
R2 := R2 OR R3;  IOCMEM(R1) := R2;
    GOTO BUFFER2;
  END;
  BEGIN                               COMMENT TESTBIT   17;
    IF R6 > 1 THEN
    BEGIN
      OPCODE := 10; GOTO STARTIO;
    END;
    R6 := R6 SHLL 4 OR R5;            COMMENT COMBINE K & J FIELDS;
    R3 := R3 SHRL R6 AND #01;         COMMENT GET KJ BIT IN R3;
    IF MONITOR AND R3 = 1 THEN
    BEGIN  OPERAND := OPERAND + 2;  GOTO GET;  END;
    IF ¬MONITOR AND R3 = 0 THEN
    BEGIN  OPERAND := OPERAND + 2;  GOTO GET;  END;
    OPERAND := OPERAND + 1; GOTO GET;
  END;
  BEGIN                               COMMENT JUMP  20;
    IF CHAIN THEN
    BEGIN
      IF ¬CAR THEN
      BEGIN
        OPERAND := R4;               COMMENT UPDATE OPERAND;
        R1 := R6 SHLL 4 + R5 SHLL 3;  COMMENT INDEX INTO IOCMEM;
        R6 := IOCMEM(R1) AND #3F;   R4 := R4 SHLL 6 + R6;
        IOCMEM(R1) := R4;             COMMENT STORE ADDR IN CAP;
```

```
    END;
    INST := R3; GOTO GOT;
  END
  ELSE GOTO ENDCHAIN;
END;
BEGIN                           COMMENT ILLEGAL CAR OR CAP INST 21;
  R2 := ASR SHRL 20 SHLL 4;        COMMENT GET CP NO.;
  IF ¬CAR THEN
  BEGIN
    IF R6 > 15 THEN R6 := #06 ELSE R6 := #07;
    R1 := R5 SHLL 5 OR R6;
    R2 := R2 OR R1;
  END;
  R1 := 3;  INTERRUPT;
END;
BEGIN                              COMMENT LOAD IOCMEM   22;
  R6 := R6 SHLL 4 + R5 SHLL 3 + 4; COMMENT MOD KJ FIELD TO INDEX CM;
  IOCMEM(R6) := R3;
END;
REALCLOCK := R3;                   COMMENT LOAD RTC   23;
BEGIN                              COMMENT STORE IOCMEM    24;
  R6 := R6 SHLL 4 + R5 SHLL 3 + 4;
  R2 := IOCMEM(R6);
  R1 := 4;  MEMORY;                COMMENT STORE R2 IN MEM(R4);
END;
BEGIN                              COMMENT  SETBIT   25;
  IF R6 > 1 THEN
  BEGIN
    OPCODE := 10;  GOTO STARTIO;
  END;
  R6 := R6 SHLL 4 OR R5;
  R2 := 1 SHLL R6 OR R3;           COMMENT SET BIT IN (Y);
  R1 := 4;  MEMORY;                COMMENT STORE (Y) MODIFIED;
END;

BEGIN                              COMMENT CLEAR BIT  26;
  IF R6 > 1 THEN
  BEGIN OPCODE := 14;  GOTO STARTIO;  END;
  R6 := R6 SHLL 4 OR R5;           COMMENT COMBINE KJ FIELD;
  R1 := 1 SHLL R6;                 COMMENT GET 1 IN KJ BIT;
  R2 := _1 XOR R1 AND R3;
  R1 := 4; MEMORY;                 COMMENT STORE BACK IN MEM(R4);
END;
BEGIN                              COMMENT TEST & SET   27;
  IF R3 < 0 THEN                   COMMENT BIT 31 SET;
  BEGIN
    OPERAND := OPERAND + 1;        COMMENT UPDATE OPERAND;
    GOTO GET;
```

```
      END  ELSE
      BEGIN
        OPERAND :=  OPERAND + 2;       COMMENT UPDATE OPERAND;
        R2 := R3 OR #80000000;         COMMENT SET BIT 31;
        R1 := 4;  MEMORY;  GOTO GET;   COMMENT STORE OPERAND MODIFIED;
      END;
    END;
  END;                                 COMMENT END OF CASE ON OPCODE;
  IF CHAIN THEN
  BEGIN  OPERAND := OPERAND + 1;  GOTO GET;  END;
  GOTO DONE;
ILLEGALINST:  OPCODE := 10;  GOTO STARTIO;
BUFFER1:  R1 := R2;
BUFFER2:
             COMMENT R5 = J,  R1 = INDEX TO IOCMEM(1ST WORD);
  INST := R1;                COMMENT SAVE POINTER TO IOCMEM WORD IN USE;
  R8 := IOCMEM(R1);
  R6 := R8 SHRL 4 AND #03;             COMMENT K IN R6;
  OPERAND := R8 SHRL 6;                COMMENT GET ADDR OF NEXT INST;
  R8 := R8 SHLL 28;
  IF R8 < 0 THEN SET(CHAIN)
  ELSE RESET(CHAIN);
  R8 := R8 SHLL 1;
  IF R8 < 0 THEN SET(MONITOR)
  ELSE RESET(MONITOR);
  R3 := IOCMEM(INST + 4);                COMMENT R3 HAS BFA & BCA;
  R2 := R2 - R2;
  SLDL(R2,16);  R7 := R2;         COMMENT SHIFT BFA INTO R2 & PUT IN R7;
  R3 := R3 SHRL 16;
          COMMENT END OF SETUP FOR CONTROL BY IOCMEM;
  R5 := R5 + 1;                        COMMENT SCALE J FOR CASE STM;
  CASE R5 OF
  BEGIN
    NULL; NULL;  NULL;  NULL; NULL;
    BEGIN                              COMMENT CH NO 5 CARD READER;
      IF R6 = 0 THEN                   COMMENT DATA SUPPRESSION;
      BEGIN
        R5 := R7 SHLL 16 OR R7;
        IOCMEM(44) := R5;              COMMENT BFA & BCA BACK TO IOCMEM;
      END ELSE                         COMMENT DATA TRANSFER;
      BEGIN
        R0 := @CBUF;  READ;  R8 := R8 - R8;
        FOR R4 := R3 STEP 1 UNTIL R7 DO
        BEGIN
          R2 := ICBUF(R8);
          R1 := 4;  MEMORY;
          R8 := R8 + 4;
          IF R8 >= 84 THEN
```

```
        BEGIN  R0 := @CBUF;  READ;  R8 := R8 - R8;  END;
   COMMENT TEST FOR END OF FILE ?????????;
    END;
    R7 := R7 SHLL 16 OR R4;
    IOCMEM(44) := R4;              COMMENT STORE BFA & BCA IN CM;
  END;
    IF MONITOR THEN
    BEGIN
      R2 := ASR SHRL 20 SHLL 1;  R1 := INTLOCKOUT(R2) AND #20;
      IF R1 ¬= 0 THEN
      BEGIN
        R1 := 3;  R2 := R2 SHLL 8 OR #5F;  INTERRUPT;
      END;
    END;               COMMENT END FOR K > 0;
  GOTO ENDCHAIN;
  END;                             COMMENT END CASE FOR CH 5 READER;
BEGIN                              COMMENT CASE FOR CH 6 PRINTER;
  R8 := R8 - R8;
  IF R6 = 0 THEN                   COMMENT OUTPUT ZEROS;
  BEGIN
    R2 := R8;
    FOR R4 := R3 STEP 1 UNTIL R7 DO
    BEGIN
      IWBUF(R8) := R2;  R8 := R8 +4;      COMMENT ZERO WBUF;
      IF R8 >= 132 THEN
      BEGIN
        R0 := @WBUF;  WRITE;  R8 := R2;
        MVC(131,WBUF,BLANK);
      END;
    END;
    IF R8 < 132 THEN WRITSTRING;
  END ELSE
  BEGIN
    FOR R4 := R3 STEP 1 UNTIL R7 DO
    BEGIN
      R1 := 3;  MEMORY;          COMMENT R2<- MEM(R4);
      IWBUF(R8) := R2;  R8 := R8 + 4;  COMMENT LOAD WBUF;
      IF R8 >= 132 THEN
      BEGIN
        R0 := @WBUF;  WRITE;  R8 := R8 - R8;
        MVC(131,WBUF,BLANK);
      END;
    END;
  IF R8 < 132 THEN WRITSTRING;
  END;
  R7 := R7 SHLL 16 OR R4;
  IOCMEM(180) := R7;               COMMENT STORE BFA & BCA IN CM;
    IF MONITOR THEN
```

```
        BEGIN
          R2 := ASR SHRL 20 SHLL 1;  R1 := INTLOCKOUT(R2) AND #40;
          IF R1 ¬= 0 THEN
          BEGIN
            R1 := 3;  R2 := R2 SHLL 8 OR #6E;  INTERRUPT;
          END;
        END;
      GOTO ENDCHAIN;
      END;
     NULL;  NULL;  NULL;  NULL;  NULL;  NULL;  NULL;  NULL;  NULL;
    NULL;                 CCMMENT    CHANNELL 7 - 15;
    END;                                   COMMENT END CASE ON CH NO.;
ENDCHAIN:  IF CHAIN THEN
  BEGIN
    OPERAND := IOCMEM(INST) SHRL 6;      COMMENT GET CAP;
    GOTO GET;
  END;
DONE:
  LM(R0,R15,SAVIO);
END;
```

```
PROCEDURE REPEATTERM (R14);
  COMMENT THIS PROCEDURE CHECKS FOR REPEAT
  INSTRUCTION TERMINATION BASED ON SPECIAL A VALUE;
BEGIN
  INTEGER SAV14;  ARRAY 3 INTEGER REGSAVTERM;
  SAV14 := R14;  STM(R2,R4,REGSAVTERM);
  R2 := REPEATAVAL;  R2 := R2 + 1;
  IF COMPARE THEN
  BEGIN  COMMENT CHECK FCR COMPARE INSTRUCTION TERMINATION;
    CASE R2 OF
    BEGIN                              COMMENT CASE OF SPECIAL A VALUE;
      BEGIN                                              COMMENT A = 0;
        R3 := ASR AND #04;
        IF R3 = 0 THEN GOTO TERMINATE;
      END;
      BEGIN                                              COMMENT A = 1;
        R3 := ASR AND #04;
        IF R3 ¬= 0 THEN GOTO TERMINATE;
      END;
      BEGIN                                              COMMENT A = 2;
        R3 := ASR AND #06;
        IF R3 = 0 THEN GOTO TERMINATE;
      END;
      BEGIN                                              COMMENT A = 3;
        R3 := ASR AND #02;
        IF R3 ¬= 0 THEN GOTO TERMINATE;
      END;
      BEGIN                                              COMMENT A = 4;
        R3 := ASR AND #02;
        IF R3 = 0 THEN GOTO TERMINATE;
      END;
      BEGIN                                              COMMENT A = 5;
        R3 := ASR AND #06;
        IF R3  ¬= 6 OR R3 ¬= 0 THEN GOTO TERMINATE;
      END;
      BEGIN                                              COMMENT A = 6;
        R3 := ASR AND #01;
        IF R3 ¬= 0 THEN GOTO TERMINATE;
      END;
      BEGIN                                              COMMENT A = 7;
        R3 := ASR AND #01;
        IF R3 = 0 THEN GOTO TERMINATE;
      END;
    END;
  END ELSE
  BEGIN  COMMENT CHECK FOR NON-COMPARE INSTRUCTION TERMINATION;
    R3 := ASR SHLL 21;  IF R3 < 0 THEN R4 := 104 ELSE R4 := 0;
    R3 := INST AND #3800000 SHRL 21 + CPUBASE + R4;  R3 := B3;
```

```
    CASE R2 OF
    BEGIN                                COMMENT CASE OF SPECIAL A VALUE;
      BEGIN                                                COMMENT A = 0;
        IF R3 ¬= 0 THEN GOTO TERMINATE;
      END;
      BEGIN                                                COMMENT A = 1;
        IF R3 = 0 THEN GOTO TERMINATE;
      END;
      BEGIN                                                COMMENT A = 2;
        IF R3 >= 0 THEN GOTO TERMINATE;
      END;
      BEGIN                                                COMMENT A = 3;
        IF R3 < 0 THEN GOTO TERMINATE;
      END;
      NULL;                                                COMMENT A = 4;
      BEGIN                                                COMMENT A = 5;
        R2 := R2 - R2;
        FOR R4 := 0 STEP 1 UNTIL 31 DO
        BEGIN
          IF R3 < 0 THEN R2 := R2 + 1;
          R3 := R3 SHLL 1;
        END;  R2 := R2 AND #01;
        IF R2 ¬= 0 THEN GOTO TERMINATE;
      END;
      BEGIN                                                COMMENT A = 6;
        R2 := R2 - R2;
        FOR R4 := 0 STEP 1 UNTIL 31 DO
        BEGIN
          IF R3 < 0 THEN R2 := R2 + 1;
          R3 := R3 SHLL 1;
        END;  R2 := R2 AND #01;
        IF R2 = 0 THEN GOTO TERMINATE;
      END;
      NULL;                                                COMMENT A = 7;
    END;
  END;
  GOTO FINISH;
TERMINATE:RESET(REPEAT); RESET(COMPARE); RESET(REPLACE);
FINISH:LM(R2,R4,REGSAVTERM);  R14 := SAV14;
END;
```

```
PROCEDURE DUMP (R14);
BEGIN COMMENT THIS PROCEDURE DUMPS MEMORY AND CMR REGISTERS AS
SPECIFIED BY MEMORY DUMP CONTROL CARD AND REGISTER DUMP CONTROL CARD;

   INTEGER SAVE14;  ARRAY  13 INTEGER REGDUM;
   SAVE14 := R14;  STM(R0,R12,REGDUM);
   R0 := @WBUF;  WRITE;
   R1 := MDUMP(4);  IF R1 = _1 THEN GOTO WREG;
   MVC(20,WBUF,"ADDRESS   MEMORY DUMP");  WRITSTRING;
   MVC(20,WBUF,"DECIMAL   HEXIDECIMAL");  WRITSTRING;
   R1 := 3;  R3 := R3 - R3;  R11 := #0F;  R12 := 1;  R8 := R3;
WMEM: R7 := MDUMP(R3);  IF R7 = _1 THEN GOTO WREG;  R5 := MDUMP(R3+4);
   MVC(131,WBUF,BLANK);  WRITE;
   FOR R4 := R7 STEP 1 UNTIL R5 DO
   BEGIN
      CVD(R4,SAVEX);  UNPK(5,7,WBUF,SAVEX);  SETZONE(WBUF(5));
      FOR R9 := 17 STEP 17 UNTIL 116 DO
      BEGIN
         MEMORY;
         R10 := R9 - 7;
         FOR R9 := R9 STEP _1 UNTIL R10 DO
         BEGIN
            R6 := R2 AND R11;
            IC(R8,HEX(R6));
            STC(R8,WBUF(R9));
            R2 := R2 SHRL 4;
         END;
        R4 := R4 + R12;
         IF R4 > R5 THEN
         BEGIN
            WRITSTRING;  GOTO OUT;
         END;
      END;
      WRITSTRING;  R4 := R4 - 1;
   END;
OUT:R3 := R3 + 8;  IF R3 <= 40 THEN GOTO WMEM;
WREG: R2 := R2 - R2;  IC(R2,NCPU);  R3 :=240;  R2 := R2 - 1 SHLL 2;
   FOR R1 := 0 STEP 4 UNTIL R2 DO
   BEGIN
      R0 := @WBUF;  WRITE;
      MVC(9,WBUF(15),"CPU NUMBER");  STC(R3,WBUF(26));
      WRITSTRING;
      CPUBASE := CPUCM(R1);  DUMPREGS;  R3 := R3 + 1;
   END;
   LM(R0,R12,REGDUM);  R14 := SAVE14;
END;
```

```
SEGMENT PROCEDURE INITIALIZ2 (R14);
BEGIN
  INTEGER SAVE14;  SAVE14 := R14;
  RO := @WBUF; WRITE;
  MVC(28,WBUF,"INITIALIZATION PHASE COMPLETE");  WRITE;
  MVC(28,WBUF,BLANK);  WRITE;
  MVC(36,WBUF,"AN/UYK-7 CPU AND MEMORY CONFIGURATION");  WRITE;
  MVC(36,WBUF,BLANK);  WRITE;
  MVC(29,WBUF,"THE NUMBER OF CPU'S ACTIVE ARE");
  IC(R2,NCPU);  R2 := R2 OR #F0:  STC(R2,WBUF(31));
  WRITE:  MVC(31,WBUF,BLANK);
  MVC(30,WBUF,"THE NUMBER OF PAGES IN CORE ARE");
  R2 := PPAGES:  CVD(R2,SAVEX);  UNPK(2,7,WBUF(32),SAVEX);
  SETZONE(WBUF(34));  WRITE;  MVC(34,WBUF,BLANK);
  IF PAGING THEN
  BEGIN
   MVC(30,WBUF,"THE NUMBER OF VIRTUAL PAGES ARE");
    R3 := VPAGES;  CVD(R3,SAVEX);  UNPK(2,7,WBUF(32),SAVEX);
    SETZONE(WBUF(34));  WRITE;  MVC(34,WBUF,BLANK);
  END;
  MVC(28,WBUF,"MEMORY AVAILABLE IS FROM 0 TO");
  R2 := MEMLIMIT -1;  CVD(R2,SAVEX);  UNPK(5,7,WBUF(30),SAVEX);
  SETZONE(WBUF(35));  WRITE:  MVC(35,WBUF,BLANK);
  R1 := R1 - R1;  R2 := R1;  IC(R1,NCPU);  R3 := #F0;
  FOR R4 := 1 STEP 1 UNTIL R1 DO
  BEGIN
    R5 := CPUCM(R2);  R6 := MEM(R5 + 328);  R7 := MEM(R5 + 100);
    R7 := R7 AND #800;  IF R7 = 0 THEN R7 := 64 ELSE R7 := 168;
    R8 := R6 AND #E0000 SHRL 15;  R8 := R8 + R7 + R5;  R8 := B8;
    R6 := R6 AND #FFFF + R8 AND #3FFFF;
    MVC(49,WBUF,"PROGRAM EXECUTION FOR CPU   WILL START AT LOCATION");
    STC(R3,WBUF(26));  R3 := R3 + 1;  CVD(R6,SAVEX);
    UNPK(5,7,WBUF(51),SAVEX);  SETZONE(WBUF(56));  WRITE;
    MVC(57,WBUF,BLANK);  R2 := R2 + 4;
  END;
  FOR R2 := 0 STEP 8 UNTIL 40 DO
  BEGIN
    R3 := MDUMP(R2);  R4 := MDUMP(R2+4);
    IF R3 = -1 THEN  GOTO NEXTD;
    MVC(28,WBUF,"MEMORY DUMP IS FROM        TO");
    CVD(R3,SAVEX);  UNPK(5,7,WBUF(20),SAVEX);  SETZONE(WBUF(25));
    CVD(R4,SAVEX);  UNPK(5,7,WBUF(30),SAVEX);  SETZONE(WBUF(35));
    WRITE;  MVC(35,WBUF,BLANK);
  END;
NEXTD:IF TRACEDO THEN
  BEGIN
    R3 := #F0;  R2 := R2 - R2;
    FOR R4 := 1 STEP 1 UNTIL R1 DO
```

```
      BEGIN
        R5 := TRACEVAL(R2);  R6 := TRACEVAL(R2 + 4);
        MVC(32,WBUF,"TRACE FOR CPU   IS FROM        TO");
        CVD(R5,SAVEX);  UNPK(5,7,WBUF(24),SAVEX);  SETZONE(WBUF(29));
        CVD(R6,SAVEX);  UNPK(5,7,WBUF(34),SAVEX);  SETZONE(WBUF(39));
        STC(R3,WBUF(14));  R3 := R3 + 1;  R2 := R2 + 8;
        WRITE;  MVC(41,WBUF,BLANK);
      END;
    END ELSE
    BEGIN
      MVC(21,WBUF,"NO TRACE IS TO BE DONE");  WRITE;
      MVC(21,WBUF,BLANK);
    END;
    MVC(131,WBUF,BLANK);  WRITE;  WRITE;
    MVC(24,WBUF,"**** START EXECUTION ****");  WRITE;
    MVC(131,WBUF,BLANK);  WRITE;
    R14 := SAVE14;
  END;                            COMMENT END INITIALIZ2;
```

```
SEGMENT PROCEDURE INITIALIZE (R14);
  COMMENT THIS PROCEDURE INITIALIZES AN/UYK-7 MEMORY AND CMR REGISTERS;
BEGIN    INTEGER SAVE14;
   SAVE14 := R14;
   MVC(16,WBUF,"    CONTROL CARDS");  WRITSTRING;
   R5 := 1;
READ1: R0 := @CBUF;   READ;
   IF > THEN GOTO EOF;  OI(#F0,CBUF(2));  PACK(7,2,SAVEX,CBUF);
   MVC(2,WBUF,CBUF);  WRITSTRING;  CVB(R1,SAVEX);
   R5 := R5 + R5;
   IF R5 = 2 THEN
   BEGIN  COMMENT VPAGES = # OF VIRTUAL PAGES OF MEMORY;
          COMMENT MEMLIMIT = MAX AN/UYK-7 ADDRESS + 1;
     IF R1 > 256 THEN R1 := 256;  VPAGES := R1;
     R1 := R1 SHLL 10;  MEMLIMIT := R1;  GOTO READ1;
   END;
   IF R1 < 1 OR R1 > 3 THEN
   BEGIN
     MVC(48,WBUF,"NUMBER OF CPU'S SPECIFIED INCORRECT - JOB FLUSHED");
     WRITSTRING;  R1 := 3;  BRANCH;
   END;
   STC(R1,NCPU);  R3 := R1 - 1 SHLL 1;  R14 := 4;
   FOR R4 := 0 STEP 2 UNTIL R3 DO
            COMMENT SET CPU STATE STARTING WITH CPU 0;
     CPUSTATE(R4) := R14;
   R0 := VPAGES;  GETFRAME;  PPAGES := R0;  MEMBASE := R1;
   COMMENT NOW SET UP PAGING FLAG;
   R2 := VPAGES;  R0 := PPAGES;
   IF R0 >= R2 THEN COMMENT IF PHYS PAGES >= VIRT PAGES THEN PAGING=FALS;
     RESET(PAGING)
   ELSE
   BEGIN  COMMENT SET UP FOR PAGING;
     SET(PAGING);  R0 := MEMBASE;  R3 := R2 - 1;
     FOR R4 := 0 STEP 1 UNTIL R3 DO
       INITPAGE;
     R0 := _1;  INITPAGE;  R4 := #E000;  R5 := PPAGES - 1 SHLL 1;
     R6 := 1;
     FOR R3 := 0 STEP 2 UNTIL R5 DO
     BEGIN    COMMENT MARK FIRST SET OF PAGES IN CORE;
       PAGETABLE(R3) := R4;  R4 := R4 + R6;
     END;
   END;
   R0 := @CBUF;  READ;
   IF > THEN GOTO EOF;  R2 := R2 - R2;  IC(R2,CBUF);  R2:= R2 SHLL 8;
   IC(R2,CBUF(1));  MVC(79,WBUF,CBUF);  WRITSTRING;
   IF R2 ¬= "MD" THEN
   BEGIN
     MVC(44,WBUF,"NO MEMORY CONTROL CARD SUPPLIED - JOB FLUSHED");
```

```
  WRITSTRING;  R1 := 3;  BRANCH;
 END:  R3 := R3 - R3;  R4 := R3;  R5 := R3;  R14 := @CBUF;
 FOR R1 := 3 STEP 7 UNTIL 74 DO
 BEGIN
  R12 := R14 + R1;
  IC(R5,CBUF(R1+5));  IF R5 = #40 THEN GOTO NEXT;
  PACK(7,5,SAVEX,B12);  CVB(R2,SAVEX);
  MDUMP(R3) := R2;  R4 := R4 XOR _1;  R1 := R1 + R4;
  R3 := R3 + 4;
 END;
NEXT: R6 := MEMLIMIT;  R7 := _1;
 FOR R2 := 0 STEP 8 UNTIL 40 DO
 BEGIN
  R3 := MDUMP(R2);  R4 := MDUMP(R2 + 4);
  IF R3 = R7 THEN GOTO NEXTA;
  IF R3 >= R6 OR R4 >= R6 THEN
  BEGIN
   MDUMP(R2) := R7;  GOTO WSTA;
  END;
  IF R3 ¬= R7 AND R4 = R7 THEN
  BEGIN
   MDUMP(R2 + 4) := R3;  GOTO WSTA;
  END;
 END;
 GOTO NEXTA;
WSTA:MVC(60,WBUF,"ADDRESS DISAGREEMENT ON MEMORY CONTROL CARD PARTIAL DU
MP DONE");  WRITSTRING;
NEXTA:R0 := @CBUF;  READ;
 IF > THEN GOTO EOF;  R2 := R2 - R2;  IC(R2,CBUF);
 R2 := R2 SHLL 8;  IC(R2,CBUF(1));  MVC(79,WBUF,CBUF);  WRITSTRING;
 IF R2 ¬= "RD" THEN
 BEGIN
  MVC(43,WBUF,"NO REGISTER DUMP CARD SUPPLIED - JOB FLUSHED");
  R1 := 3;  BRANCH;
 END;
 R2 := R2 - R2;  R3 := R2;  R5 := R2;
 IC(R2,CBUF(12));  IC(R3,CBUF(15));  IC(R5,CBUF(18));
 IF R2 = "I" THEN SET(ECHOI) ELSE RESET(ECHOI);
 IF R3 = "L" THEN SET(ECHOL) ELSE RESET(ECHOL);
 IF R5 = "1" THEN SET(FORMAT1) ELSE RESET(FORMAT1);
 R2 := R2 - R2;  R3 := R2;  R5 := R2;  STARTDUMP := R2;
 R4 := 130;  ENDDUMP := R4;  R6 := 3;  R7 := #40;
OCT:IC(R5,CBUF(R6));  IC(R3,CBUF(R6 + 1));  IC(R2,CBUF(R6 + 2));
 IF R2 = R7 THEN GOTO WSTB ELSE R2 := R2 AND #0F;
 IF R3 = R7 THEN R3 := R3 - R3 ELSE R3 := R3 AND #0F SHLL 3;
 IF R5 = R7 THEN R5 := R5 - R5 ELSE R5 := R5 AND #0F SHLL 6;
 R2 := R2 + R3  + R5;
 IF R6 = 3 THEN
```

```
  BEGIN
    STARTDUMP := R2;  R6 := R6 + 4;   GOTO OCT;
  END;
  R7 := STARTDUMP;
  IF R2 > 130 THEN R2 := 130;
  IF R7 > R2 THEN
  BEGIN
    R7 := R7 - R7;  STARTDUMP := R7;  GOTO WSTB;
  END;
  ENDDUMP := R2;
  GOTO NEXTB;
WSTB:MVC(62,WBUF,"ADDRESS DISAGREEMENT ON REGISTER CONTROL CARD PARTIAL
DUMP DONE");  WRITSTRING;
NEXTB:RO := @CBUF;  READ;
  IF > THEN GOTO EOF;  R2 := R2 - R2;  IC(R2,CBUF);
  MVC(79,WBUF,CBUF);  WRITSTRING;
  IF R2 ¬= "T" THEN
  BEGIN
    MVC(35,WBUF,"NO TRACE CARD SUPPLIED - JOB FLUSHED");
    WRITSTRING;  R1 := 3;  BRANCH;
  END;  IC(R2,CBUF(1));
  IF R2 = "N" THEN
  BEGIN
    RESET(TRACEDO);  GOTO NEXTC;
  END ELSE  SET(TRACEDO);  IC(R2,NCPU);  R2 := R2 SHLL 1;
  R3 := @CBUF;  R4 := 2;
  FOR R5 := 1 STEP 1 UNTIL R2 DO
  BEGIN
    R6 := R3 + R4;  R7 := R4 + 5;  R8 := R8 - R8;  IC(R8,CBUF(R7));
    IF R8 = #40 THEN
    BEGIN
      MVC(32,WBUF,"ERROR ON TRACE CARD - JOB FLUSHED");
      WRITSTRING;  R1 := 3;  BRANCH;
    END;  PACK(7,5,SAVEX,B6);  CVB(R1,SAVEX);  R10 := R5 - 1 SHLL 2;
    TRACEVAL(R10) := R1;  R4 := R4 + 7;
  END;
NEXTC:RO := @CBUF;  READ;
  IF > THEN GOTO EOF;  R2 := R2 - R2;  IC(R2,CBUF);
  IF R2 ¬= "%" THEN
  BEGIN
    MVC(47,WBUF,"ERROR IN CONTROL CARDS NO PERCENT SIGN FOLLOWING");
    WRITSTRING;  R1 := 3;  BRANCH;
  END;
  IF ¬FORMAT1 THEN GOTO ASSEM;
COMMENT START READING INSTRUCTION CARDS AT THIS POINT;
    RO := @WBUF;  WRITE;
    MVC(23,WBUF,"OCTAL INSTRUCTION FORMAT");  WRITE;
    MVC(23,WBUF,BLANK);
```

```
    R8 := MEMLIMIT;  R4 := 0;  R10 := #OF;
IREAD:  RO := @CBUF;  READ;
    IF > THEN GOTO EOF;  R12 := R12 - R12;
    IF ECHOI THEN
    BEGIN
      MVC(79,WBUF(8),CBUF);
    END;
    R1 := R12;  IC(R1,CBUF);
    IF R1 = "%" THEN GOTO ENDINST;
    R1 := 7;  R9 := 4;  R2 := R2 - R2;
HALF: FOR R3 := R12 STEP 1 UNTIL R9 DO
    BEGIN
      IC(R5,CBUF(R3));  R5 := R5 AND R1;  R2 := R2 SHLL 3 OR R5;
    END;
    IC(R5,CBUF(R3));  R5 := R5 AND 1;  R2 := R2 SHLL 1 OR R5;
    IF HWFLAG THEN BEGIN RESET(HWFLAG);  GOTO STORE;  END;
    R3 := R3 + 1;
    IC(R5,CBUF(R3));  R5 := R5 AND R1;  R2 := R2 SHLL 3 OR R5;
    IF R2 >= #60000 THEN GOTO SECHW;
    R7 := R12;
    FOR R3 := 7 STEP 1 UNTIL 10 DO
    BEGIN
      R5 := R12;  IC(R5,CBUF(R3));
      R5 := R5 AND R10;
      R7 := R7 * 10 + R5;
    END;
    R2 := R2 SHLL 13 OR R7;
STORE: R3 := R12;  IC(R3,CBUF(19));
    IF R3 = #40 THEN GOTO TESTLIM;
    PACK(7,5,SAVEX,CBUF(14));
    CVB(R1,SAVEX);
    R4 := R1;       COMMENT ADDRESS IN CARD;
TESTLIM:  IF ECHOI THEN
    BEGIN
      CVD(R4,SAVEX);  UNPK(5,7,WBUF,SAVEX);  SETZONE(WBUF(5));
      WRITSTRING;
    END;
    IF R4 <  R8 THEN
    BEGIN
      R1 := 4;  MEMORY;  R4 := R4 + 1;
    END ELSE
    BEGIN
      IF ¬ECHOI THEN
      BEGIN
        MVC(79,WBUF,CBUF);  RO := @WBUF;  WRITE;
        MVC(79,WBUF,BLANK);
      END;
      MVC(32,WBUF,"ABOVE ADDRESS BEYOND MEMORY LIMIT ");
```

```
      RO := @WBUF;  WRITE;  MVC(32,WBUF,BLANK);  R1 := 3;  BRANCH;
    END;
    GOTO IREAD;
SECHW: R12 := 7;  R9 := 10;  SET(HWFLAG);  GOTO HALF;
  COMMENT START READING INSTRUCTIONS ACCORDING TO ASSEMBLER FORMAT;
ASSEM:RO := @WBUF;  WRITE;
  MVC(27,WBUF,"ASSEMBLER INSTRUCTION FORMAT");  WRITE;
  MVC(27,WBUF,BLANK);
  R12 := R12 - R12;  R11 := MEMLIMIT;
IRD:RO := @CBUF;  READ;
  IF > THEN GOTO EOF;  R4 := R12;  IC(R4,CBUF);
  IF R4 = "%" THEN GOTO ENDINST;  R5 := R12;  R6 := R12;
  R4 := R4 SHLL 16;  IC(R5,CBUF(1));  R5 := R5 SHLL 8;
  IC(R6,CBUF(2));  R4 := R4 OR R5 OR R6 AND #3FFFF;  R5 := R12;
  IC(R5,CBUF(3));  IF R5 = 215 THEN GOTO PREG;
  IF R5 > 19 THEN GOTO ERRI;  R6 := @CBUF;
  FOR R7 := 1 STEP 1 UNTIL R5 DO
  BEGIN
    R6 := R6 + 4;
    MVC(3,IAW,B6);  R2 := IAW;
    IF R4 >= R11 THEN GOTO ERRI;
    R1 := 4;  MEMORY;
    R4 := R4 + 1;
  END;
  GOTO IRD;
PREG:MVC(3,IAW,CBUF(4));  R2 := IAW AND #EFFFF;
  MVC(3,IAW,CBUF(8));  R3 := IAW AND #EFFFF;
  MVC(3,IAW,CBUF(12));  R4 := IAW AND #EFFFF;
  R5 := CPUCM + 328;  R6 := R12;  IC(R6,NCPU);
  B5 := R2;  R5 := CPUCM(4) + 328;
  IF R6 > 1 THEN B5 := R3;
  R5 := CPUCM(8) + 328;
  IF R6 > 2 THEN B5 := R4;
  GOTO ENDINST;
ERRI:MVC(79,WBUF,CBUF);  RO := @WBUF;  WRITE;
  MVC(79,WBUF,BLANK);
  MVC(79,WBUF,"LOAD ADDRESS FOR ASSEMBLER FORMAT GREATER THAN AVAILABLE
IN MEMORY - JOB FLUSHED");  WRITE;  MVC(79,WBUF,BLANK);
  R1 := 3;  BRANCH;
ENDINST: WRITSTRING;
  COMMENT COLUMN 1 CONTAINS A INDICATOR AS TO A REGISTER VALUE OR DATA
  VALUE BY EITHER A "R" OR "D",  COLUMN 3 - 8 CONTAINS THE ADDRESS,
  COLUMN 10 - 17 CONTAINS THE VALUE, A CARD WITH "%" MUST BE PROVIDED;
WRITSTRING;
  MVC(27,WBUF,"REGISTER/DATA INITIALIZATION");  WRITSTRING;
  R12 := 57;  R11 := 4;  R10 := 240;  R9 := #0F;
  R8 := MEMLIMIT;  R7 := CPUCM(0);
AREAD:RO :=@CBUF;  READ;
```

```
  IF > THEN GOTO EOF;  R1 := R1 - R1;  IC(R1,CBUF);
  IF ECHOL THEN
  BEGIN
    MVC(79,WBUF,CBUF);  RO := @WBUF;  WRITE;  MVC(79,WBUF,BLANK);
  END;
  IF R1 = "%" THEN GOTO FINISHLOAD;
ERR:IF R1 = #40 THEN
  BEGIN
    MVC(79,WBUF,CBUF);  RO := @WBUF;  WRITE;  MVC(79,WBUF,BLANK);
    MVC(51,WBUF,"ERROR IN ABOVE CARD ON LOADING REGS/DATA-JOB FLUSHED");
    WRITSTRING;  R1 := 3;  BRANCH;
  END;
  R2 := R2 - R2;  R3 := R2;  IC(R2,CBUF(7));  IC(R3,CBUF(16));
  IF R2 = #40 OR R3 = #40 THEN
  BEGIN  R1 := #40;  GOTO ERR;  END;
  PACK(7,5,SAVEX,CBUF(2));  CVB(R4,SAVEX);  R2 := R2 - R2;
  FOR R3 := 9 STEP 1 UNTIL 16 DO
  BEGIN
    R5 := R5 - R5;  IC(R5,CBUF(R3));
    IF R5 < R10 THEN  R5 := R5 + R12;
    R5 := R5 AND R9;  R2 := R2 SHLL R11 OR R5;
  END;
  IF R4 > 255 AND R1 = "R" THEN
  BEGIN  R1 := #40;  GOTO ERR;  END;
  IF R4 >= R8 AND R1 = "D" THEN
  BEGIN  R1 := #40;  GOTO ERR;  END;
  IF R1 = "D" THEN
  BEGIN  R1 := 4;  MEMORY;  GOTO AREAD;  END;
  IF R1 = "R" THEN
  BEGIN
    IF R4 = 25 OR R4 = 110 OR R4 = 195 THEN
    BEGIN
      R4 := R4 SHLL 2 + R7;  R3 := B4 AND #700000;
      R2 := R2 AND #FFFFF;  R3 := R3 OR R2;  B4 := R3;
    END ELSE
    BEGIN
      R4 := R4 SHLL 2 + R7;  B4 := R2;
    END;  GOTO AREAD;
  END;
EOF:MVC(34,WBUF,"END-OF-FILE ON READER - JOB FLUSHED");
  WRITSTRING;  R1 := 3;  BRANCH;
FINISHLOAD:INITIALIZ2;
  R14 := SAVE14;
END;  COMMENT END PROCEDURE INITIALIZE;
```

```
SEGMENT PROCEDURE WTRACE (R14);
BEGIN  COMMENT THIS PROCEDURE OUTPUTS THE TRACE;


PROCEDURE WHEX (R14);
BEGIN
  R3 := #0F;
  FOR R4 := 0 STEP 4 UNTIL 28 DO
  BEGIN
    R5 := R2 SHRL R4 AND R3;  IC(R5,HEX(R5));
    STC(R5,WBUF(R1));  R1 := R1 - 1;
  END;
END;


  INTEGER SAV14;  ARRAY 11 INTEGER REGWT;
  SAV14 := R14;  STM(R0,R10,REGWT);
  R0 := @WBUF;  WRITE;
  MVC(23,WBUF,"UYK-7 ADDRESS (DECIMAL):");  R1 := LOC;
  CVD(R1,SAVEX);  UNPK(5,7,WBUF(25),SAVEX);  SETZONE(WBUF(30));
  R1 := ASR SHRL 20 AND #07 OR #F0;
  IF EXECSTATE THEN
  BEGIN
    MVC(27,WBUF(33),"IN INTERRUPT STATE FOR CPU #");
    STC(R1,WBUF(62));
  END ELSE
  BEGIN
    MVC(22,WBUF(33),"IN TASK STATE FOR CPU #");
    STC(R1,WBUF(57));
  END;  WRITE;  MVC(63,WBUF,BLANK);
  MVC(31,WBUF,"ACTIVE STATUS REGISTER (IN HEX):");
  MVC(19,WBUF(43),"P REGISTER (IN HEX):");
  R2 := ASR;  R1 := 40;  WHEX;
  R2 := PREGVAL;  R1 := 71;  WHEX;  WRITE;  MVC(75,WBUF,BLANK);
  MVC(7,WBUF,"OP CODE:");  R1 := OPCODE AND #07 OR #F0;
  STC(R1,WBUF(10));  R1 := OPCODE SHRL 3 OR #F0;
  STC(R1,WBUF(9));  MVC(22,WBUF(13),"ACCUMULATOR DESIGNATOR:");
  IF OPCODE < 48 THEN
    R1 := R7 SHRL 2 OR #F0
  ELSE
    R1 := REGWT(16) AND #380 SHRL 7 OR #F0;  STC(R1,WBUF(37));
  MVC(28,WBUF(40),"ACCUMULATOR A VALUE (IN HEX):");
  IF OPCODE > 47 THEN R2 := B7
  ELSE  BEGIN R2 := REGWT(20);  R2 := B2;  END;
  R1 := 77;  WHEX;  WRITE;  MVC(79,WBUF,BLANK);
  IF OPCODE > 47 THEN GOTO HALF;
  MVC(16,WBUF,"INDEX DESIGNATOR:");  R2 := INST SHRL 17 AND #07 OR #F0;
  STC(R2,WBUF(18));
```

```
  IF INDEX THEN
  BEGIN
    R2 := INDEXREGLOC;  R2 := B2;
    MVC(29,WBUF(21),"INDEX REGISTER VALUE (IN HEX):");
    R1 := 59;  WHEX;
  END ELSE
    MVC(10,WBUF(21),"NO INDEXING");
  WRITE;  MVC(60,WBUF,BLANK);
  IF IDIR THEN
    MVC(21,WBUF,"INDIRECT ADDRESSING ON")
  ELSE
    MVC(21,WBUF,"NO INDIRECT ADDRESSING");
  MVC(24,WBUF(19),"BASE REG DESIGNATOR:");
  R1 := INST SHRL 13 AND #07 OR #F0;  STC(R1,WBUF(45));
  MVC(24,WBUF(48),"BASE REG VALUE (DECIMAL):");
  R1 := BASEREGLOC;  R1 := B1;  CVD(R1,SAVEX);
  UNPK(5,7,WBUF(74),SAVEX);  SETZONE(WBUF(79));  WRITE;
  MVC(91,WBUF,BLANK);  MVC(31,WBUF,"BASIC OPERAND ADDRESS (DECIMAL):");
  R1 := INST AND #1FFF;  CVD(R1,SAVEX);
  UNPK(3,7,WBUF(33),SAVEX);  SETZONE(WBUF(36));
  MVC(32,WBUF(39),"FINAL COMPUTED ADDRESS (DECIMAL):");
  R1 := OPERAND;  CVD(R1,SAVEX);  UNPK(5,7,WBUF(73),SAVEX);
  SETZONE(WBUF(78));  WRITE;  MVC(80,WBUF,BLANK);
  IF CHARADDR THEN
  BEGIN
    MVC(34,WBUF,"CHARACTER ADDRESSING  IAW (IN HEX):");
    R2 := IAW;  R1 := 43;  WHEX;
    MVC(34,WBUF(46),"IAWP (DECIMAL):     IAWW (DECIMAL):");
    R1 := IAWP;  CVD(R1,SAVEX);  UNPK(1,7,WBUF(62),SAVEX);
    R1 := IAWW;  CVD(R1,SAVEX);  UNPK(1,7,WBUF(82),SAVEX);
    SETZONE(WBUF(63));  SETZONE(WBUF(83));  WRITE;
    MVC(90,WBUF,BLANK);  GOTO FINCA;
  END;
  IF IDIR THEN
  BEGIN
    MVC(37,WBUF,"NO CHARACTER ADDRESSING  IAW (IN HEX):");
    R2 := IAW;  R1 := 46;  WHEX;  WRITE;  MVC(50,WBUF,BLANK);
  END;
FINCA:IF OPCODE < 8 THEN GOTO F2;
  IF OPCODE < 40 THEN GOTO F1;
  IF OPCODE > 43 THEN GOTO F1;
  MVC(13,WBUF,"F3 DESIGNATOR:");  R1 := R6 SHRL 1 OR #F0;
  STC(R1,WBUF(15));
  IF OPCODE = 40 THEN
  BEGIN
    MVC(30,WBUF(18),"ACCUMULATOR A+1 VALUE (IN HEX):");
    R2 := REGWT(20) + ANEXT;  R2 := B2;  R1 := 57;  WHEX;
  END;  WRITE;  MVC(81,WBUF,BLANK);  GOTO FINISH;
```

```
F2:  MVC(13,WBUF,"F2 DESIGNATOR:");  R1 := R6 OR #F0;
  STC(R1,WBUF(15));
  IF OPCODE < 7 THEN
  BEGIN
    MVC(12,WBUF(19),"(Y) (IN HEX):");
    R1 := 1;  R6:= OPERAND;  MEMORY;  R2 := OPERAND;
    R1 := 40;  WHEX;  OPERAND := R6;
  END ELSE
  BEGIN
    IF R6 = 6 THEN
    BEGIN
      R1 := CPUBASE + INDEXG + 28;  R1 := B1;
      MVC(18,WBUF(19),"B7 VALUE (DECIMAL):");
      CVD(R1,SAVEX);  UNPK(5,7,WBUF(39),SAVEX);  SETZONE(WBUF(44));
    END;
  END;  WRITE;  MVC(79,WBUF,BLANK);
  MVC(30,WBUF,"ACCUMULATOR A+1 VALUE (IN HEX):");
  R2 := REGWT(20) + ANEXT;  R2 := B2;  R1 := 39;  WHEX;
  IF OPCODE = 5 OR OPCODE = 6 THEN
  BEGIN
    MVC(14,WBUF(43),"(Y+1) (IN HEX):");
    R1 := 1;  R6 := OPERAND;  OPERAND := OPERAND + 1;  MEMORY;
    R2 := OPERAND;  R1 := 66;  WHEX;  OPERAND := R6;
  END;  WRITE;  MVC(70,WBUF,BLANK);  GOTO FINISH;
F1:MVC(12,WBUF,"K DESIGNATOR:");  R1 := R6 OR #F0;
  STC(R1,WBUF(14));  MVC(12,WBUF(17),"(Y) (IN HEX):");
  R1 := 1;  R6 := OPERAND;  MEMORY;  R2 := OPERAND;  OPERAND := R6;
  R1 := 38;  WHEX;  WRITE;  MVC(40,WBUF,BLANK);
  MVC(30,WBUF,"ACCUMULATOR A+1 VALUE (IN HEX):");
  R2 := REGWT(20) + ANEXT;  R2 := B2;  R1 := 39;  WHEX;
  IF OPCODE > 15 AND OPCODE < 20 THEN
  BEGIN
    R2 := REGWT(28);  IF R2 ¬= 0 THEN
    BEGIN
      MVC(13,WBUF(42),"B(A) (IN HEX):");
      R2 := R2 + CPUBASE + INDEXG;  R2 := B2;
      R1 := 64;  WHEX;
    END;
  END;  WRITE;  MVC(65,WBUF,BLANK);  GOTO FINISH;
HALF:IF OPCODE > 49 AND OPCODE < 56 THEN
  BEGIN
    MVC(21,WBUF,"SHIFT AMOUNT (IN HEX):");
    R2 := REGWT(16) AND #7F;  R1 := 30;  WHEX;
    MVC(30,WBUF(33),"ACCUMULATOR A+1 VALUE (IN HEX):");
    R2 := REGWT(28) + ANEXT;  R1 := 72;  R2 := B2;  WHEX;
    WRITE;  MVC(79,WBUF,BLANK);  GOTO FINISH;
  END ELSE
  BEGIN
```

```
    MVC(50,WBUF,"F4 DESIGNATOR:    INDEX DESIGNATOR:    I DESIGNATOR");
    R1 := REGWT(16);  R2 := R1 SHRL 4 AND #07 OR #F0;
    STC(R2,WBUF(15));  R2 := R1 SHRL 1 AND #07 OR #F0;
    STC(R2,WBUF(36));  R2 := R1 AND #01;
    IF R2 > 0 THEN MVC(1,WBUF(52),"ON") ELSE MVC(2,WBUF(52),"OFF");
    WRITE:  MVC(60,WBUF,BLANK);
    IF OPCODE > 55 AND OPCODE < 60 THEN
    BEGIN
      MVC(30,WBUF,"ACCUMULATOR A+1 VALUE (IN HEX):");
      R2 := REGWT(28) + ANEXT;  R2 := B2;  R1 := 39;  WHEX;
      MVC(19,WBUF(42),"A(B) VALUE (IN HEX):");
      R2 := REGWT(20);  R1 := 70;  WHEX;
      WRITE:  MVC(79,WBUF,BLANK);
    END;
  END;
FINISH:LM(R0,R10,REGWT);  R14 := SAV14;
END;
```

```
SEGMENT PROCEDURE HALFWORD (R14);
BEGIN
  INTEGER SAVE14;


PROCEDURE SHIFTAMT (R14);  COMMENT OBTAINS THE SHIFT AMOUNT FOR
                    INSTRUCTIONS 62 - 67 AND RETURNS VALUE IN R6;
BEGIN
  INTEGER SAV14;  ARRAY 3 INTEGER REGSAVESH;
  SAV14 := R14;  STM(R3,R5,REGSAVESH);
  R3 := R4 AND #40;
  IF R3 = 0 THEN
  BEGIN  COMMENT SHIFT AMOUNT IN M FIELD;
    R6 := R4 AND #3F;
  END ELSE
  BEGIN
    R3 := R4 AND #20;
    IF R3 = 0 THEN
    BEGIN  COMMENT SHIFT IN B(B);
      R4 := R4 SHRL 1 AND #07;
      IF R4 = 0 THEN  COMMENT NOOP;
      BEGIN  R8 := 18;  GOTO ENDCASE;  END;
      R5 := R5 + 32;  R6 := B5 AND #3F;
    END ELSE
    BEGIN  R6 := B5 AND #3F;  END;
  END;
  LM(R3,R5,REGSAVESH);  R14 := SAV14;
END;


  SAVE14 := R14;
EXECHALF:  R4 := ASR AND #8000;
  IF R4 = 0 THEN R4 := INST SHRL 16 ELSE R4 := INST AND #FFFF;
  OPCODE := R4 SHRL 10;
  IF OPCODE = 0 THEN BEGIN R8 := 15;  GOTO ENDCASE;  END;
  R7 := R4 AND #380 SHRL 5;  R6 := ASR SHLL 21;
  R3 := ASR AND #F0000;
  IF R3 = 0 THEN RESET(EXECSTATE) ELSE SET(EXECSTATE);
  R5 := R4 AND #0E SHLL 1;
  IF R7 = 28 THEN R3 := _28 ELSE R3 := 4;  ANEXT := R3;
  IF R6 >= 0 THEN
  BEGIN
    R7 := R7 + CPUBASE;  COMMENT A(A) ADDRESS: TASK STATE;
    R5 := R5 + CPUBASE;  COMMENT A(B) ADDRESS : TASK STATE;
  END ELSE
  BEGIN
    R7 := R7 + CPUBASE + 104;  COMMENT A(A) ADDRESS: INTERRUPT STATE;
    R5 := R5 + CPUBASE + 104;  COMMENT A(B) ADDRESS: INTERRUPT STATE;
```

```
  END;
  IF OPCODE > 49 AND OPCODE < 56 THEN
    GOTO ADJUSTOPCODE   COMMENT FORMAT IVB INSTRUCTION;
  ELSE
  BEGIN  COMMENT FORMAT IVA INSTRUCTION;
    R6 := R4 SHLL 25 SHRL 29;  COMMENT SUBFUNCTION CODE DESIGNATOR;
    R3 := R4 AND #01;  COMMENT CHECK FOR I DESIGNATOR SET OR CLEARED;
    IF R3 = 1 THEN SET(IDESIGN) ELSE RESET(IDESIGN);
  END;
ADJUSTOPCODE:IF TRACE THEN WTRACE;
  OPCODE := OPCODE - 47;
  IF OPCODE < 1 THEN GOTO ILLEGALINST;
  CASE OPCODE OF
  BEGIN                                      COMMENT START OF CASE STATEMENT;
    BEGIN                                                COMMENT OPCODE= 60;
      R7 := R4 SHLL 22 SHRL 29;
      IF ¬EXECSTATE THEN
      BEGIN
        IF IDESIGN OR R7 > 1 THEN GOTO PRIVINST;
      END;
      R7 := R7 SHLL 3 OR R6 AND #3F;  R3 := R7;  R4 := R4 - R4;
      IF IDESIGN THEN R7 := R7 + 64;  IC(R4,CMRADDR(R7));
      IF R4 = 255 THEN GOTO ILLEGALINST;  R4 := R4 SHLL 2;
      R2 := CPUBASE + R4;  R2 := B2;
      IF R3 > 8 AND R3 < 16 THEN
        R2 := R2 AND #FFFF;  B5 := R2;
      R8 := 18;
    END;
    BEGIN                                              COMMENT OPCODE = 61;
      R7 := R4 SHLL 22 SHRL 29;
      IF ¬EXECSTATE THEN
      BEGIN
        IF IDESIGN OR R7 > 1 THEN GOTO PRIVINST;
      END;
      R7 := R7 SHLL 3 OR R6 AND #3F;  R3 := R7;  R4 := R4 - R4;
      IF IDESIGN THEN R7 := R7 + 64;  IC(R4,CMRADDR(R7));
      IF R4 = 255 THEN GOTO ILLEGALINST;  R4 := R4 SHLL 2;
      R2 := R4 + CPUBASE;  R5 := B5;
      IF R3 > 8 AND R3 < 16 THEN
      BEGIN
        R3 := B2 AND #E0000;  R5 := R5 AND #FFFF OR R3;
      END;
      IF R4 = 100 THEN
      BEGIN
        R3 := B2 AND #7F8000;  R5 := R5 AND #7FFF OR R3;
      END;  B2 := R5;
      R8 := 18;
    END;
```

```
    BEGIN                                                   COMMENT OPCODE = 62;
      SHIFTAMT;  R6 := R6 AND #1F;  R3 := B7;  R2 := R2 - R2;
      SLDL(R2,B6);  R3 := R3 OR R2;  B7 := R3;
      R8 := 18;
    END;
    BEGIN                                                   COMMENT OPCODE = 63;
      SHIFTAMT;
      IF R6 >= 32 THEN
      BEGIN
        R3 := B7;  R7 := R7 + ANEXT;  R5 := B7;  R6 := R6 - 32;
      END ELSE
      BEGIN
        R5 := B7;  R7 := R7 + ANEXT;  R3 := B7;
      END;
      R2 := R2 - R2;  R4 := R2;
      SLDL(R2,B6);  SLDL(R4,B6);  R3 := R3 OR R4;  R5 := R5 OR R2;
      B7 := R3;  R7 := R7 - ANEXT;  B7 := R5;
      R8 := 18;
    END;
    BEGIN                                                   COMMENT OPCODE = 64;
      SHIFTAMT;  IF R6 > 31 THEN BEGIN R5 := R5 - R5;  GOTO OP64;  END;
      R5 := B7 SHRL R6;
      B7 := R5;
OP64: R8 := 18;
    END;
    BEGIN                                                   COMMENT OPCODE = 65;
      SHIFTAMT;
      R1 := R7 + ANEXT;  R2 := B1; R3 := B7;
      SRDL(R2,B6);  B1 := R2;  B7 := R3;
      R8 := 18;
    END;
    BEGIN                                                   COMMENT OPCODE = 66;
      SHIFTAMT;  IF R6 > 31 THEN R6 := 31;
      R5 := B7 SHRA R6;  B7 := R5;
      R8 := 18;
    END;
    BEGIN                                                   COMMENT OPCODE = 67;
      SHIFTAMT;
      R1 := R7 + ANEXT;  R2 := B1; R3 := B7;
      SRDA(R2,B6);  B1 := R2;  B7 := R3;
      R8 := 18;
    END;
    BEGIN                                           COMMENT OPCODE GROUP 70X;
      R6 := R6 + 1;
      CASE R6 OF
      BEGIN
        BEGIN                                              COMMENT OPCODE = 700;
          R3 := B7;
```

```
          IF R3 = 0 OR R3 = _1 THEN
          BEGIN
            IF R7 = R5 THEN GOTO OP7001;
            OPCODE := 31;  B5 := OPCODE;  GOTO OP7001;
          END;
          R2 := R2 - R2;  R4 := R2;
OP700:    SLDL(R2,1);
          R4 := R4 + 1;
          IF R3 > 0 AND R2 = 0 THEN GOTO OP700;
          IF R3 < 0 AND R2 > 0 THEN GOTO OP700;
          R4 := R4 - 1;
          SRDL(R2,1);
          R3 := R3 OR R2;
          B7 := R3;
          IF R7 ¬= R5 THEN B5 := R4;
OP7001:   R8 := 22;
        END;
        BEGIN                                             COMMENT OPCODE = 701;
          R8 := R7;  R1 := R1 - R1;
          R3 := B7;  R7 := R7 + ANEXT;  R2 := B7;
          IF R2 = 0 AND R3 = 0 THEN
          BEGIN
            IF R7 = R5 OR R8 = R5 THEN GOTO TIME701;
            OPCODE := 63;  B5 := OPCODE;
            GOTO TIME701;
          END;
          IF R2 = _1 AND R3 = _1 THEN
          BEGIN
            IF R7 = R5 OR R8 = R5 THEN GOTO TIME701;
            OPCODE := 63;  B5 := OPCODE;
            GOTO TIME701;
          END;
          IF R2 < 0 THEN R10 := 1 ELSE R10 := R10 - R10;
OP7011:   SLDL(R2,1);
          R1 := R1 + 1;
          IF R2 >= 0 AND R10 = 0 THEN GOTO OP7011;
          IF R2 < 0 AND R10 = 1 THEN GOTO OP7011;
          R1 := R1 - 1;
          SRDL(R2,1);  IF R10 = 1 THEN R2 := R2 OR #80000000;
          IF R5 = R7 OR R5 = R8 THEN GOTO OP7012;
          B5 := R1;
OP7012:   B8 := R3;  B7 := R2;
TIME701:  R8 := 22;
        END;
        BEGIN                                             COMMENT OPCODE = 702;
          R6 := B7 XOR _1;  B7 := R6;  R8 := 11;
        END;
        BEGIN                                             COMMENT OPCODE = 703;
```

```
   R2 := B7 XOR _1;  R8 := R7 + ANEXT;
   R3 := B8 XOR _1;  B7 := R2;  B8 := R3;  R8 := 11;
  END;
  GOTO ILLEGALINST;                        COMMENT OPCODE = 704;
  GOTO ILLEGALINST;                        COMMENT OPCODE = 705;
  GOTO ILLEGALINST;                        COMMENT OPCODE = 706;
  GOTO ILLEGALINST;                        COMMENT OPCODE = 707;
 END;             COMMENT END CASE STATEMENT;
END;                         COMMENT END GROUP OPCODE = 70X;
BEGIN                          COMMENT GROUP OPCODE = 71X;
 R6 := R6 + 1;
 CASE R6 OF
 BEGIN
  BEGIN                                    COMMENT OPCODE = 710;
   R3 := B5;  R2 := B7 OR R3;  B7 := R2;
  END;
  BEGIN                                    COMMENT OPCODE = 711;
   R2 := B7;  R3 := B5;  ADD;  B7 := R6;
  END;
  BEGIN                                    COMMENT OPCODE = 712;
   R2 := B7;  R3 := B5 XOR _1;  ADD;  B7 := R6;
  END;
  BEGIN                                    COMMENT OPCODE = 713;
   R3 := B5;  R2 := B7 XOR R3;  B7 := R2;
  END;
  GOTO ILLEGALINST;                        COMMENT OPCODE = 714;
  BEGIN                                    COMMENT OPCODE = 715;
   R3 := B5;  R2 := B7 AND R3;  B7 := R2;
  END;
  GOTO ILLEGALINST;                        COMMENT OPCODE = 716;
  GOTO ILLEGALINST;                        COMMENT OPCODE = 717;
 END;        COMMENT END CASE STATEMENT;
 R8 := 10;
END;                         COMMENT END GROUP CODE  =  71X;
GOTO ILLEGALINST;              COMMENT ILLEGAL OPCODE = 72X;
GOTO ILLEGALINST;              COMMENT ILLEGAL OPCODE = 73X;
BEGIN                            COMMENT GROUP OPCODE = 74X;
 R6 := R6 + 1;
 CASE R6 OF
 BEGIN
  BEGIN                                    COMMENT OPCODE = 740;
   R1 := R1 - R1;  R3 := B7;  R2 := B5;
   IF R3 < 0 THEN BEGIN R3 := R3 XOR _1;  R1 := 1;  END;
   IF R2 < 0 THEN BEGIN  R2 := R2 XOR _1;  R1 := R1 - 1;  END;
   R3 := R3 * R2;
   IF R1 ¬= 0 THEN
   BEGIN  R2:= R2 XOR _1;  R3 := R3 XOR _1;  END;
   B7 := R3;  R7 := R7 + ANEXT;  B7 := R2;
```

```
          R8 := 78;
        END;
        BEGIN                                      COMMENT OPCODE = 741;
          R1 := 0;   R8 := R1;   R3 := B7;
          R7 := R7 + ANEXT;    R2 := B7;
          IF R2 < 0 THEN
          BEGIN
            R2 := R2 XOR _1;   R3 := R3 XOR _1;   R1 := R1 + 1;
            R8 := 1;
          END;  COMMENT R1 WILL INDICATE + OR - QUOTIENT, R8 + OR - REM;
          R4 := B5;   IF R4 = 0 OR R4 = _1 THEN
          BEGIN ASR := ASR AND #FFFFFFF8;  GOTO OP741; END;
          IF R4 < 0 THEN
          BEGIN
            R4 := R4 XOR _1;  R1 := R1 - 1;
          END;
          R3 := R3 / R4;
          IF R1 ¬= 0 THEN R3 := R3 XOR _1;
          IF R8 = 1 THEN R2 := R2 XOR _1;
          B7 := R2;  R7 := R7 - ANEXT;  B7 := R3;
OP741:    R8 := 150;
        END;
        BEGIN                                      COMMENT OPCODE = 742;
          R14 := R4 AND #0E;
          IF R14 = #0E THEN R14 := _28 ELSE R14 := 4;
          ABNEXT := R14;
          R14 := R9;  COMMENT SAVE INSTRUCTION WORD;
          R1 := 1;   R8 := R8 - R8;  R3 := B7;  R7 := R7 + ANEXT;
          R2 := B7;
          IF R2 < 0 OR R2 >= #40000000 THEN
          BEGIN
            ASR := ASR OR #08;  R9 := R14;  GOTO TIME742;
          END;
          IF R2 = 0 THEN
          BEGIN
            IF R3 = 0 THEN GOTO STORESQRT;
            R8 := 31;  SLDL(R2,31);
          END;
OP7420:   SLDL(R2,1);  R8 := R8 + R1;
          IF R2 > 0 THEN GOTO OP7420;
          SRDL(R2,B8);  COMMENT RESTORE A(A), A(A+1);
          R7 := 63 - R8 SHRL 1;
          R6 := R1 SHLL R7;  COMMENT R6 = 1ST APPROX.;
          R10 := R6 SHLL 1;
          R4 := R6 SHRL 1;
          COMMENT R6 AND R10 WILL HOLD LOWER AND UPPER LIMITS;
          COMMENT R4 WILL HOLD MASK;
          R8 := R8 AND 1;  IF R8 = 0 THEN R7 := R10 ELSE R7 := R6;
```

```
OP7421:    R9 := R7 * R7;
           IF R8 > R2 THEN BEGIN R10 := R7;  GOTO OP7424;  END;
           IF R8 = R2 THEN
           BEGIN
             IF R3 = R9 THEN
             BEGIN R2 := R2 - R2;  R3 := R7;  GOTO STORESQRT;  END;
             CLR(R3,R9);  IF > THEN GOTO OP7423;
             R10 := R7;  GOTO OP7424;
           END;
OP7423:    R6 := R7;
OP7424:    R8 := R10 - R6;
           IF R8 = 0 OR R8 = 1 THEN
           BEGIN
             R9 := R6 * R6;  R2 := R3 - R9;  R3 := R6;  GOTO STORESQRT;
           END;
           R7 := R6 OR R4;  R4 := R4 SHRL 1;
           GOTO OP7421;
STORESQRT:B5 := R3;  R5 := R5 + ABNEXT;  B5 := R2;  R9 := R14;
TIME742:   R8 := 150;
         END;
         BEGIN                                         COMMENT OPCODE = 743;
           R2 := R4 AND #380;
           IF R2 = 0 THEN GOTO OP743;
           R2 := R4 AND #0E;
           IF R2 = 0 THEN
           BEGIN  R4 := R4 - R4;  GOTO OP7431;  END;
           R2 := R5 + 32;  COMMENT DISPLACE FROM ACCUM TO INDEX GROUP;
           R4 := B2 AND #FFFF;
OP7431:    R3 := R7 + 32;
           R5 := B3 AND #E0000 OR R4;
           B3 := R5;
OP743:     R8 := 18;
         END;
         BEGIN                                         COMMENT OPCODE = 744;
           ASR := ASR AND #7FFFF9;
           R2 := B5;  R3 := B7;  COMMENT R2 = A(B), R3 = A(A);
           IF R3 = R2 THEN ASR := ASR OR #06;
           IF R3 > R2 THEN ASR := ASR OR #02;
           R8 := 11;
         END;
         BEGIN                                         COMMENT OPCODE = 745;
           R2 := B7;  R7 := R7 + ANEXT;  R4 := B7;
           R3 := B5;
           IF R3 >= R2 AND R3 < R4 THEN
           BEGIN
             ASR := ASR AND #7FFFFE;  GOTO OP745;
           END ELSE ASR := ASR OR 1;
OP745:   R8 := 18;
```

```
        END;
        BEGIN                                       COMMENT OPCODE = 746;
          R2 := B7;  R7 := R7 + ANEXT;  R3 := B7;
          R4 := B5;  R3 := R3 AND R2;
          COMMENT R3 = LOGICAL PRODUCT OF A(A+1) AND A(A), R4 = A(B);
          ASR := ASR AND #7FFFF9;
          IF R3 > R4 THEN
          BEGIN
            ASR := ASR OR #02;  GOTO OP746;
          END;
          IF R3 = R4 THEN ASR := ASR OR #06;
OP746:  R8 := 11;
        END;
        BEGIN                                       COMMENT OPCODE = 747;
          R5 := R5 + 32;  R7 := R7 + 32;
          R2 := R4 AND #0E;  R3 := R4 AND #380;
          IF R2 ¬= 0 THEN R2 := B5 AND #FFFF;
          IF R2 ¬= 0 THEN R3 := B7 AND #FFFF;
          ASR := ASR AND #7FFFF9;   COMMENT CD = 00, (< CASE);
          IF R2 = R3 THEN BEGIN ASR := ASR OR #06;  GOTO OP747; END;
          IF R2 > R3 THEN ASR := ASR OR #02;
OP747:  R8 := 20;
        END;
      END;            COMMENT END CASE STATEMENT;
    END;                                  COMMENT END GROUP OPCODE = 74X;
    GOTO ILLEGALINST;                                COMMENT OPCODE = 75;
    GOTO ILLEGALINST;                                COMMENT OPCODE = 76;
    BEGIN                                     COMMENT GROUP OPCODE = 77X;
      R6 := R6 + 1;
      CASE R6 OF
      BEGIN                              COMMENT START OF CASE STATEMENT;
        BEGIN                                       COMMENT OPCODE = 770;
          COMMENT A FIELD NOT USED DUE TO ONLY ONE IOC;
          IF ¬EXECSTATE THEN GOTO PRIVINST;
          R3 := IMONCLOCK;  B5 := R3;
          R8 := 30;
        END;
        BEGIN                                       COMMENT OPCODE = 771;
          COMMENT A FIELD NOT USED DUE TO ONLY ONE IOC;
          R3 := REALCLOCK;  B5 := R3;
          R8 := 35;
        END;
        GOTO ILLEGALINST;                           COMMENT OPCODE = 772;
        GOTO ILLEGALINST;                           COMMENT OPCODE = 773;
        BEGIN                                       COMMENT OPCODE = 774;
          IF ¬EXECSTATE THEN GOTO PRIVINST;
          ASR := ASR OR #1000;
          R8 := 22;
```

```
        END;
        BEGIN                                      COMMENT OPCODE = 775;
          IF ¬EXECSTATE THEN GOTO PRIVINST;
          ASR := ASR AND #EFEFFF;
          R8 := 22;
        END;
        BEGIN                                      COMMENT OPCODE = 776;
          IF ¬EXECSTATE THEN GOTO PRIVINST;
          R8 := 22;  TIME;
          R5 := ASR SHLL 9 SHRL 29 SHLL 1;  COMMENT CP NUMBER * 2;
          IF IDESIGN THEN R1 := 2      COMMENT WAIT;
          ELSE  R1 := 1;               COMMENT HALT;
          CPUSTATE(R5) := R1;          COMMENT STORE THE CP'S STATE;
          R8 := ASR AND #8000;
          IF R8 = 0 THEN
          BEGIN
            ASR := ASR OR #8000;  COMMENT SET UPPER/LOWER BIT ON;
            R8 := MEM(CPUBASE + 328);    COMMENT BACK UP P REG BY 1;
            R6 := R8 - 1 AND #FFFF;
            R8 := R8 AND #F0000 OR R6;
            MEM(CPUBASE + 328) := R8;
          END;
          R1 := 2;  BRANCH;
        END;
      END;                               COMMENT END GROUP OPCODE = 77X;
    END;                                     COMMENT END CASE STATEMENT;
  END;                                      COMMENT END CASE STATEMENT;
ENDCASE:TIME;  R4 := ASR AND #8000;
IF R4 > 0 THEN BEGIN  ASR := ASR AND #FFFF7FFF;  GOTO FINISH;  END;
    ASR := ASR OR #8000;  GOTO EXECHALF;
ILLEGALINST:R1 := 2;  R2 := R1;  INTERRUPT;
PRIVINST:R1 := 2;  R2 := 3;  INTERRUPT;
FINISH:MEM(CPUBASE + 100) := ASR;  R14 := SAVE14;
END;                                    COMMENT END HALFWORD PROCEDURE;
```

```
SEGMENT PROCEDURE FULLWORD (R14);
BEGIN
INTEGER SAVE14;
SAVE14 := R14;
  CASE OPCODE OF
  BEGIN                                 COMMENT START OF CASE STATEMENT;
    BEGIN                                          COMMENT OPCODE = 01X;
      MEMORY;  R4 := B5;  R6 := R6 + R1;
      CASE R6 OF
      BEGIN                                COMMENT CASE OPCODE 010 - 017;
        BEGIN                                      COMMENT OPCODE = 010;
          R4 := R4 OR OPERAND;  B5 := R4;
        END;
        BEGIN                                      COMMENT OPCODE = 011;
          OPERAND := OPERAND XOR _1;  R4 := R4 AND OPERAND; B5:= R4;
        END;
        BEGIN                                      COMMENT OPCODE = 012;
          R5 := R5 + ANEXT;  R3 := B5;  R2 := R4 AND OPERAND;
          R4 := R4 XOR _1 AND R3 OR R2;  B5 := R4;
        END;
        BEGIN                                      COMMENT OPCODE = 013;
          R4 := R4 XOR OPERAND; B5:= R4;
        END;
        BEGIN                                      COMMENT OPCODE = 014;
          R5 := R5 + ANEXT;  R2 := B5;  R3 := R4 AND OPERAND;
          ADD;  B5 := R6;
        END;
        BEGIN                                      COMMENT OPCODE = 015;
          R4 := R4 AND OPERAND;  B5 := R4;
        END;
        BEGIN                                      COMMENT OPCODE = 016;
          R5 := R5 + ANEXT;  R2 := B5;  R3 := R4 AND OPERAND;
          R3 := R3 XOR _1;  ADD;  B5 := R6;
        END;
        BEGIN                                      COMMENT OPCODE = 017;
          R5 := R5 + ANEXT;  R4 := R4 AND OPERAND;  B5 := R4;
        END;
      END;                             COMMENT END CASE OPCODE 010 - 017;
      R8 := 15;
    END;
    BEGIN                                   COMMENT GROUP OPCODE = 02X;
      R4 := B5;  R6 := R6 + R1;
OP025:CASE R6 OF
      BEGIN                                 COMMENT CASE OPCODE 020 -027;
        BEGIN                                      COMMENT OPCODE = 020;
          MEMORY;  R3 := 0;
          BEGIN
            IF OPERAND < 0 THEN R3 := R3 + R1;
```

```
        OPERAND := OPERAND SHLL 1;
      END;  B5 := R3;  R8 := 75;
    END;
    GOTO ILLEGALINST;                         COMMENT OPCODE = 021;
    BEGIN                                     COMMENT OPCODE = 022;
      IF CHARADDR THEN GOTO ILLEGALINST;
      R8 := 15;  TIME;
      MEMORY;  INST := OPERAND;  OPCODE := INST SHRL 26;
      IF OPCODE >= #30 THEN
      BEGIN  INST := INST SHRL 16; ASR := ASR OR #8000;
        HALFWORD;  R8 := 0;  GOTO ENDCASE;
      END;
      R1 := 1;  BRANCH;
    END;
    BEGIN                                     COMMENT OPCODE = 023;
      IF CHARADDR THEN GOTO ILLEGALINST;
      MEMORY;  INST := OPERAND;  ASR := ASR OR #8000;
      OPCODE := INST AND #FC00 SHRL 10;
      IF OPCODE ¬= 0 THEN HALFWORD;
      R8 := 15;
    END;
    BEGIN                                     COMMENT OPCODE = 024;
      R5 := R5 + ANEXT;  R3 := B5;       R2 := R3 AND R4;
      R1 := 2;  MEMORY;
      R8 := 15;
    END;
    BEGIN                                     COMMENT OPCODE = 025;
      R5 := R5 + ANEXT;  R2 := B5;  R3 := R4;
      ADD: B5 := R6; R2 := R6; R1 := 2; MEMORY;
      R8 := 20;
    END;
    BEGIN                                     COMMENT OPCODE = 026;
      R4 := R4 XOR _1;  R6 := 6; GOTO OP025;
    END;
    BEGIN                                     COMMENT OPCODE = 027;
      IF CHARADDR THEN GOTO ILLEGALINST;
      R2 := B5;  R1 := 2;  MEMORY;
      OPERAND := OPERAND + 1;  R5 := R5 + ANEXT;  R2 := B5; MEMORY;
      R8 := 30;
    END;
  END;                              COMMENT END CASE OPCODE 020 - 027;
END;
BEGIN                                   COMMENT GROUP OPCODE = 03X;
  R4 := B5;
  IF REPLACE AND R6 < 7 THEN    COMMENT MODIFY STORE ADDRESS FOR
  INSTRUCTION IN THE REPEAT MODE BY USING S(6) INSTEAD OF S(S);
    R7 := REPLACESTOREVAL
  ELSE
```

```
    R7 := OPERAND;  MEMORY;  R6 := R6 + 1;
  CASE R6 OF
  BEGIN                                 COMMENT CASE OPCODE 030 - 037;
    BEGIN                                        COMMENT OPCODE = 030;
      R2 := R4 OR OPERAND;  OPERAND := R7;  R1 := 2;
      MEMORY;  B5 := R2;
    END;
    BEGIN                                        COMMENT OPCODE = 031;
      R2:= OPERAND XOR _1;  R2 := R2 AND R4;  B5 := R2;
      OPERAND := R7;  R1 := 2;  MEMORY;
    END;
    BEGIN                                        COMMENT OPCODE = 032;
      R5 := R5 + ANEXT;  R6 := B5;  R1 := R4 AND OPERAND;
      R2 := R4 XOR _1 AND R6 OR R1;
      OPERAND := R7;  B5 := R2;  R1 := 2; MEMORY;
    END;
    BEGIN                                        COMMENT OPCODE = 033;
      IF CHARADDR THEN GOTO ILLEGALINST;
      R2 := R4 XOR OPERAND;  B5 := R2;  OPERAND := R7;
      R1 := 2;  MEMORY;
    END;
    BEGIN                                        COMMENT OPCODE = 034;
      R5 := R5 + ANEXT;  R3 := R4 AND OPERAND;  R2 := B5;  ADD;
      B5 := R6;  R2 := R6;
      OPERAND := R7;  R1 := 2; MEMORY;
    END;
    BEGIN                                        COMMENT OPCODE = 035;
      R5 := R5 + ANEXT;  R2 := R4 AND OPERAND;  OPERAND := R7;
      R1 := 2;  B5 := R2;  MEMORY;
    END;
    BEGIN                                        COMMENT OPCODE = 036;
      R5 := R5 + ANEXT;  R2 := B5;  R3 := OPERAND AND R4;
      R3 := R3 XOR _1;  ADD;  R2 := R6;
      B5 := R6;  OPERAND := R7;  R1 := 2; MEMORY;
    END;
    BEGIN                                        COMMENT OPCODE = 037;
      IF CHARADDR THEN GOTO ILLEGALINST;
      IF OPERAND < 0 THEN ASR := ASR AND #FFFFFFFB ELSE
      BEGIN
        ASR := ASR OR #06;
        R2 := OPERAND OR #80000000;
        OPERAND := R7;  R1 := 2;  MEMORY;
      END;
    END;
  END;                                COMMENT END CASE OPCODE 030 - 037;
  R8 := 25;
END;
GOTO ILLEGALINST;                               COMMENT GROUP OPCODE = 04X;
```

```
    BEGIN                                   COMMENT GROUP OPCODE = 05X;
      IF CHARADDR THEN GOTO ILLEGALINST;
      IF R6 > 4 THEN  GOTO ILLEGALINST;
      R4 := B5;  R6 := R6 + R1;
OP051:CASE R6 OF
      BEGIN                               COMMENT CASE OPCODE 050 - 054;
        BEGIN                                     COMMENT OPCODE = 050;
          R6 := OPERAND;  MEMORY;  B5 := OPERAND;
          OPERAND := R6 + R1;  R5 := R5 + ANEXT;
          MEMORY;  B5 := OPERAND;
          R8 := 30;
        END;
        BEGIN                                      COMMENT OPCODE = 051;
          R3 := OPERAND;  MEMORY;  ASR := ASR AND #FFFFFFF7;
          R2 := OPERAND;  OPERAND := R3 + R1;
          MEMORY;
          IF OPCODE = _1 THEN
          BEGIN
            OPERAND := OPERAND XOR _1;
            R2 := R2 XOR _1;
          END;
          ALR(R2,R4);
          IF > OR OVERFLOW THEN R4 := R1 ELSE R4 := R4 - R4;
          R5 := R5 + ANEXT;
          R4 := R4 + B5;
          R3 := R4;
          R6 := R6 - R6;
          ALR(R3,OPERAND);
          IF OVERFLOW OR > THEN R6 := R1;
          IF R4 < 0 AND OPERAND < 0 AND R3 >= 0 THEN
            ASR := ASR OR #08 ELSE
          IF R4 > 0 AND OPERAND > 0 AND R3 < 0 THEN
            ASR := ASR OR #08;
          R2 := R2 + R6;
          B5 := R3;  R5 := R5 - ANEXT;  B5 := R2;
          R8 := 30;
        END;
        BEGIN  R6 := 2;  OPCODE := _1;            COMMENT OPCODE = 052;
          GOTO OP051;
        END;
        BEGIN                                     COMMENT OPCODE = 053;
          R5 := R5 + ANEXT;  R6 := OPERAND;  OPERAND := OPERAND + R1;
          MEMORY;  R4 := B5;
          IF R4 > OPERAND THEN R2 := #02;
          IF R4 = OPERAND THEN GOTO EQTEST;
          IF R4 < OPERAND THEN R2 := R2 - R2;
SET12:    ASR := ASR AND #FFFFFFF9 OR R2;
          GOTO TIME053;
```

```
EQTEST:   R5 := R5 - ANEXT;  R3 := B5;  OPERAND := R6;
          MEMORY;  CLR(R3,OPERAND);
          IF = THEN
          BEGIN
            R2 := #06;  GOTO SET12;
          END;
          IF < THEN
          BEGIN
            R2 := #00;  GOTO SET12;
          END;
          R2 := #02;  GOTO SET12;
TIME053:  R8 := 30;
        END;
        BEGIN                                       COMMENT OPCODE = 054;
          IF ¬EXECSTATE THEN
          BEGIN
            R4 := ASR AND #100;  IF R4 = 0 THEN GOTO PRIVINST;
            R4 := INST SHLL 16 SHRL 29;
            IF R4 ¬= 7 THEN GOTO PRIVINST;
            R4 := INST SHLL 6 SHRL 29;
            IF R4 = 7 THEN  GOTO PRIVINST;
          END;
          R3 := INST AND #1FFF + INDEXREGVAL AND #FFFF;
          R4 := R3 SHLL 31;
          IF R4 < 0 THEN  GOTO ILLEGALINST;
          R5 := INST AND #E000 SHLL 4 OR R3;
          R6 := CPUBASE + R7 +296;  B6 := R5;
          R4 := OPERAND + 1;  MEMORY;  R8 := OPERAND;  OPERAND := R4;
          MEMORY;  R5 := CPUBASE + R7 +64;  R8 := R8 AND #3FFFF;
          B5 := R8;  OPERAND := OPERAND AND #1FFFFF;
          R5 := CPUBASE + R7 + 264;  B5 := OPERAND;
          R8 := 58;
        END;
      END;                             COMMENT END CASE OPCODE 051 - 054;
    END;
    BEGIN                                      COMMENT OPCODE GROUP 06X;
      IF CHARADDR THEN GOTO ILLEGALINST;
      R2 := B5;  R4 := OPERAND + R1;  MEMORY;
      R3 := OPERAND;  OPERAND := R4;  R8 := R5 + ANEXT;
      MEMORY;  R4 := OPERAND;  R5 := B8;
      COMMENT R2 = A(A), R3 = Y, R4 = Y + 1, R5 = A(A+1), R8 = ADDR A+1;
      IF R2 > 32767 OR R2 <_32768 THEN GOTO OP06X2;
      IF R3 > 32767 OR R3 <_32768 THEN GOTO OP06X2;
      IF R4 > #40000000 OR R4 < #BFFFFFFF OR R4 = 0 OR R4 = _1 THEN
        GOTO OP06X1 ELSE GOTO OP06X2;
OP06X1:IF R5 >=#40000000 OR R5 <=#BFFFFFFF OR R5 = 0 OR R5 = _1 THEN
        GOTO OP06X3;
OP06X2:R1 := 2;  R2 := 1;  INTERRUPT;
```

```
OP06X3:R6 := R6 + R1;
OP06: CASE R6 OF
      BEGIN                               COMMENT CASE OPCODE OF 060 - 067;
        BEGIN                                       COMMENT OPCODE = 060;
          R3 := R3 XOR _1;  ADD;  R3 := R3 XOR _1;
          IF R6 > 30 THEN GOTO TIME060;  COMMENT A(A) MUCH GREATER;
          IF R6 < _31 THEN
          BEGIN COMMENT Y MUCH GREATER SO Y, Y+1 INTO A(A), A(A+1);
            B8 := R4;  R8 := R8 - ANEXT;  B8 := R3;  GOTO TIME060;
          END;
          IF R6 < 0 THEN  COMMENT A(A) LESS THAN Y;
          BEGIN
            OPERAND := R3;  COMMENT OPERAND = LARGER CHARACTERISTIC;
            R7 := R6 XOR _1;  COMMENT R7 = AMOUNT TO SHIFT SMALLER;
            R2 := R5 SHRA R7;  COMMENT SHIFT A(A+1) WHICH IS SMALLER;
            R3 := R4;          COMMENT  R3 = Y+1;
          END ELSE
          BEGIN            COMMENT A(A) >= Y;
            OPERAND := R2;  R2 := R5;  R3 := R4 SHRA R6;
          END;
          INST := ASR;    COMMENT SAVE A COPY OF ASR;
          ADD;  ASR := ASR AND #08;
          IF ASR > 0 THEN      COMMENT OVERFLOW FROM LAST ADD;
          BEGIN    COMMENT R6 NOW CONTAINS RESULT OF MANTISSA ADD;
            IF R6 >= 0 THEN
            BEGIN
              R6 := R6 SHRL 1 OR #80000000;
              OPERAND := OPERAND + 1;
            END ELSE
            BEGIN R6 := R6 SHRL 1;  OPERAND := OPERAND + 1;  END;
          END ELSE
          BEGIN   COMMENT NORMALIZE MANTISSA IN R6;
            IF R6 = 0 OR R6 = _1 THEN GOTO OP0600;
            R1 := 0;
            IF R6 < 0 THEN
            BEGIN
              WHILE R1 > R6 DO   COMMENT NORMALIZE NEG MANTISSA;
              BEGIN
                R6 := R6 SHLL 1 OR 1;  OPERAND := OPERAND - 1;
              END;
              R6 := R6 SHRL 1 OR #80000000;  OPERAND := OPERAND + 1;
            END ELSE
            BEGIN
              WHILE R1 < R6 DO   COMMENT NORMALIZE POS MANTISSA;
              BEGIN
                R6 := R6 SHLL 1;  OPERAND := OPERAND - 1;
              END;
              R6 := R6 SHRL 1;  OPERAND := OPERAND + 1;
```

```
            END;
          END;
OP0600:   ASR := INST;    COMMENT RESTORE ASR;
          B8 := R6;    COMMENT STORE MANTISSA IN A(A+1);
          R8 := R8 - ANEXT;
          B8 := OPERAND;  COMMENT STORE CHARACTERISTIC IN A(A);
          IF OPERAND > 32767 OR OPERAND < _32768 THEN
            GOTO OP06X2;
TIME060:  R8 := 62;
        END;
        BEGIN                                            COMMENT OPCODE = 061;
          R4 := R4 XOR _1;  R6 := 1;  GOTO OP06;
        END;
        BEGIN                                            COMMENT OPCODE = 062;
          ADD;  OPERAND := R6;
          R1 := 0;
          IF R4 < 0 THEN BEGIN R4 := R4 XOR _1;  R1 := 1;  END;
          IF R5 < 0 THEN BEGIN R5 := R5 XOR _1;  R1 := R1 - 1;  END;
          IF R5 = 0 OR R4 = 0 THEN
          BEGIN
            R5 := R5 - R5;  OPERAND := R5;  GOTO STORE061;
          END;
          R5 := R5 * R4;
          WHILE R4 > 0 DO
          BEGIN
            SLDL(R4,1);  OPERAND := OPERAND - 1;
          END;
          R4 := R4 SHRL 1;  OPERAND := OPERAND + 2;
          IF R1 ¬= 0 THEN BEGIN R4 := R4 XOR _1;  END;
STORE061: B8 := R4;  R8 := R8 - ANEXT;  B8 := OPERAND;
          IF OPERAND > 32767 OR OPERAND < _32768 THEN
            GOTO OP06X2;
          R8 := 100;
        END;
        BEGIN                                            COMMENT OPCODE = 063;
          IF R4 = 0 OR R4 = _1 THEN GOTO OP06X2;
          R1 := 0;     COMMENT MAKE NUMBERS POSITIVE AND SET FLAG;
          IF R5 < 0 THEN BEGIN  R5 := R5 XOR _1;  R1 := 1;  END;
          IF R4 < 0 THEN BEGIN R4 := R4 XOR _1;  R1 := R1 - 1;  END;
          R3 := R3 XOR _1;  ADD;  R7 := R6;  COMMENT R7 = A(A)-(Y);
          R2 := R5;  R3 := R4;     COMMENT R2 = A(A+1), R3=(Y+1);
          R4 := R4 - R4;
          FOR INST := 0 STEP 1 UNTIL 31 DO
          BEGIN
            CLR(R3,R2);
            IF <= THEN
            BEGIN
              SLR(R2,R3);  R2 := R2 SHLL 1;  R4 := R4 SHLL 1 OR 1;
```

```
            END ELSE
            BEGIN
              R4 := R4 SHLL 1;  R2 := R2 SHLL 1;
            END;
          END;
          IF R4 < 0 THEN
          BEGIN      COMMENT ADJUST ANSWER IF NECESSARY;
            R4 := R4 SHRL 1;  R7 := R7 + 1;
          END;
          IF R1 ¬= 0 THEN R4 := R4 XOR _1;
          B8 := R4;  R8 := R8 - ANEXT;  B8 := R7;
          IF R7 > 32767 OR R7 < _32768 THEN GOTO OP06X2;
          R8 := 170;
        END;
        BEGIN                                      COMMENT OPCODE = 064;
          R6 := 1;  GOTO OP06;
        END;
        BEGIN                                      COMMENT OPCODE = 065;
          R6 := 1;  R4 := R4 XOR _1;  GOTO OP06;
        END;
        BEGIN                                      COMMENT OPCODE = 066;
          R6 := 3;  GOTO OP06;
        END;
        BEGIN                                      COMMENT OPCODE = 067;
          R6 := 4;  GOTO OP06;
        END;
      END;                  COMMENT END CASE;
    END;                                    COMMENT END GROUP OPCODE 06X;
  END;                                   COMMENT END CASE STATEMENT;
ENDCASE:  TIME:  MEM(CPUBASE + 100) := ASR;  GOTO FINISH;
ILLEGALINST:R1 := 2;  R2 := R1;  INTERRUPT;
PRIVINST:R1 := 2;  R2 := 3;  INTERRUPT;
FINISH:R14 := SAVE14;
END;  COMMENT END PROCEDURE FULLWORD;
```

```
SEGMENT PROCEDURE FULLWORD2 (R14);
BEGIN
INTEGER SAVE14;
SAVE14 := R14;
CASESTATE:OPCODE := OPCODE - 6;
  CASE OPCODE OF
  BEGIN                                 COMMENT START OF CASE STATEMENT;
    BEGIN                                          COMMENT OPCODE = 07X;
      IF CHARADDR THEN GOTO ILLEGALINST;
      R6 := R6 + 1;
OP07X:CASE R6 OF
      BEGIN                COMMENT CASE OF 070 - 076;
        BEGIN                                      COMMENT OPCODE = 070;
          R2 := INST AND #FFFF + INDEXREGVAL AND #FFFF;
          IF R7 = 0 THEN                COMMENT A = 0 CLASS J INTERRUPT;
          BEGIN
            R1 := 4;  INTERRUPT;
          END ELSE
          BEGIN
            IF ¬EXECSTATE THEN GOTO PRIVINST;
            R8 := 3;
            R3 := R2 SHLL 16;           COMMENT SELF-INTERRUPT IN SIGN;
            R6 := ASR SHLL 20;          COMMENT CPU NUMBER;
            FOR R4 := 0 STEP 1 UNTIL 2 DO
            BEGIN
              R9 := R4 SHLL 1;
              IF R4 = R6 AND R3 < 0 THEN CPUSTATE(R9) := R8
              ELSE
              BEGIN
                R7 := 31 - R4;  R5 := R2 SHLL R7;  R1 := CPUSTATE(R9);
                IF R5 < 0 AND R1 >= 1 THEN CPUSTATE(R9) := R8;
              END;
            END;
          END;
          R8 := 40;
        END;
        BEGIN                                      COMMENT OPCODE = 071;
          COMMENT A FIELD NOT USED DUE TO ONLY ONE IOC;
          IF ¬EXECSTATE THEN GOTO PRIVINST;
          R7 := INST AND #E0000 SHRL 15 + CPUBASE + BASEG;
          R7 := B7 AND #FFFF;
          R6 := INST AND #FFFF + R7;
          R5 := ASR AND #700000 SHRL 20 SHLL 1;
          IF OPCODE > 0 THEN
          BEGIN
            R7 := INTLOCKOUT(R5);
            R6 := R6 OR R7;  INTLOCKOUT(R5) := R6;
          END ELSE
```

```
  BEGIN
    R7 := INTLOCKOUT(R5)  OR R6;
    R6 := R6 XOR R7;  INTLOCKOUT(R5) := R6;
  END;
  R8 := 20;
END;
BEGIN                                       COMMENT OPCODE = 072;
  OPCODE := 0;  R6 := 2;  GOTO OP07X;
END;
BEGIN                                       COMMENT OPCODE = 073;
  COMMENT A FIELD NOT USED DUE TO ONLY ONE IOC;
  IF ¬EXECSTATE THEN GOTO PRIVINST;
  R7 := INST AND #E0000 SHRL 15 + CPUBASE + BASEG;
  R7 := B7 AND #FFFF;
  R6 := INST AND #FFFF + R7;
  IMONCLOCK := R6;
  IF R6 = 0 THEN
  BEGIN  COMMENT IOC-CP INTERRUPT;
    R1 := 3;  R2 := 11;  INTERRUPT;
  END;
  R8 := 30;
END;
BEGIN                                       COMMENT OPCODE = 074;
  IF ¬EXECSTATE THEN GOTO PRIVINST;
  IOC;  R8 := 35;
END;
BEGIN                                       COMMENT OPCODE = 075;
  IF ¬ EXECSTATE THEN GOTO PRIVINST;
  R7 := ASR SHLL 12;  COMMENT R7 SIGN BIT = STATE I BIT;
  IF R7 < 0 THEN
  BEGIN    COMMENT RETURNING FROM A CLASS I INTERRUPT;
    R7 := MEM(CPUBASE + 204) AND #FFFFF;
    MVC(3,MEM(R11+328),MEM(R11+212));  COMMENT RESTORE PREG;
    GOTO END075;
  END;
  R7 := R7 SHLL 1;
  IF R7 < 0 THEN
  BEGIN    COMMENT RETURNING FROM A CLASS II INTERRUPT;
    R7 := MEM(CPUBASE +220) AND #FFFFF;
    MVC(3,MEM(R11+328),MEM(R11+228));
    GOTO END075;
  END;
  R7 := R7 SHLL 1;
  IF R7 < 0 THEN
  BEGIN    COMMENT RETURNING FROM A CLASS III INTERRUPT;
    R7 := MEM(CPUBASE + 236) AND #FFFFF;
    MVC(3,MEM(R11+328),MEM(R11+244));
    GOTO END075;
```

```
          END;
          R7 := MEM(CPUBASE + 252) AND #FFFFF;
          MVC(3,MEM(R11+328),MEM(R11+260));
END075:   ASR := ASR AND #700000 OR R7;     COMMENT ASR = OLD ASR;
          R8 := 30;
        END;
        BEGIN                                        COMMENT OPCODE = 076;
          COMMENT B(7) IS CHECK TO DETERMINE WHETHER TO REPEAT THE NEXT
          INSTRUCTION OR NOT, IF SO REPEATOP ARRAY IS CHECK TO SEE IF IT
          IS ALLOWABLE, VALUES IN REPEATOP ARE: 1- NON-COMPARE,
          2- COMPARE, 3- NOT-REPEATABLE, 4- FURTHER LOOK TO DETERMINE,
          5- REPLACE, INSTRUCTION, 6- ILLEGAL INSTRUCTION;
          RESET(REPEAT);  RESET(REPLACE);  RESET(COMPARE);
          R4 := CPUBASE + INDEXG + 28;  B7ADDR := R4;  R4 := B4;
          IF R4 = 0 THEN
          BEGIN   COMMENT DO NOT REPEAT THE NEXT INSTRUCTION;
            R4 := MEM(CPUBASE + 328);  R4 := R4 + 1;
            MEM(CPUBASE +328) := R4;  GOTO TIME076;
          END ELSE
          BEGIN
            REPEATCOUNT := R4;  REPEATINST := INST;
            R1 := 0;  MEMORY;  OPCODE := INST SHRL 26;
            R6 := INST SHLL 9 SHRL 29;  R3 := R3 - R3;
            IC(R3,REPEATOP(OPCODE));
NEXT076:    CASE R3 OF
            BEGIN
              RESET(COMPARE);
              SET(COMPARE);
              GOTO TIME076;
              BEGIN
                R3 := R3 - R3;  IC(R3,REPEATOP02(R6));  GOTO NEXT076;
              END;
              BEGIN  RESET(COMPARE);  SET(REPLACE);  END;
              GOTO ILLEGALINST;
             END;
          END;
          R1 := REPEATINST;
          IF REPLACE THEN
          BEGIN  COMMENT CHECK FOR B ¬= 0 FOR REPLACE INSTRUCTION;
            R2 := R1 SHLL 12  SHRL 29;
            IF R2 = 0 THEN RESET(REPLACE);
          END;
          R2 := R1 AND #FFFF SHLL 16  SHRA 16;
          REPEATINDEX := R2;  R2 := R1 SHLL 6 SHRL 29;
          REPEATAVAL := R2;  SET(REPEAT);
          R2 := 2;  REPEATSKIP := R2;
          R2 := R1 SHRL 26;
          IF R2 > 43 AND R2 < 48 THEN
```

```
            SET(REPEATCMR)
          ELSE
            RESET(REPEATCMR);
TIME076:  R8 := 15;
        END;
        GOTO ILLEGALINST;                         COMMENT OPCODE = 077;
      END;
    END;
    BEGIN                                            COMMENT OPCODE 10;
      IF CHARADDR THEN BEGIN  MEMORY;  GOTO CA10;  END;
      IF R6 ¬= 0 THEN MEMORY;
      NORMREAD;
CA10: B5 := OPERAND;
      R8 := 15;
    END;
    BEGIN                                            COMMENT OPCODE 11;
      IF ¬ INDEX THEN GOTO TIME11;  R7 := R6;
      R4 :=  INDEXREGLOC;  R2 := B4 AND #FFFF;  R3 := R1;  ADD;
      R6 := R6 AND #FFFF;  R2 := R2 AND #E0000 OR R6;
      B4 := R2;   OPCODE := 8;  R6 := R7;  GOTO CASESTATE;
TIME11: R8 := 15;
    END;
    BEGIN                                          COMMENT OPCODE = 12;
      IF CHARADDR THEN BEGIN  MEMORY;  GOTO CA12;  END;
      IF R6 ¬= 0 THEN MEMORY;
      NORMREAD;
CA12: R2 := OPERAND;  R3 := B5 XOR _1;  ADD;
      R5 := R5 + ANEXT;  B5 := R6;
      R8 := 15;
    END;
    BEGIN                                          COMMENT OPCODE = 13;
      IF CHARADDR THEN BEGIN  MEMORY;  GOTO CA13;  END;
      IF R6 ¬= 0 THEN MEMORY;
      NORMREAD;
CA13: R2 := B5;  R3 := OPERAND XOR _1;  ADD;
      B5 := R6;
      R8 := 15;
    END;
    BEGIN                                          COMMENT OPCODE = 14;
      IF CHARADDR THEN BEGIN  MEMORY;  GOTO CA14;  END;
      IF R6 ¬= 0 THEN MEMORY;
      NORMREAD;
CA14: R2 := B5;  R3 := OPERAND;  ADD;
      B5 := R6;
      R8 := 15;
    END;
    BEGIN                                          COMMENT OPCODE = 15;
      IF CHARADDR THEN BEGIN  MEMORY;  GOTO CA15;  END;
```

```
         IF R6 ¬= 0 THEN MEMORY;
         NORMREAD;
CA15:    R2 := OPERAND;  R3 := B5;  ADD;
         R5 := R5 + ANEXT;  B5 := R6;
         R8 := 15;
      END;
      BEGIN                                              COMMENT OPCODE = 16;
         IF CHARADDR THEN BEGIN  MEMORY;  GOTO CA16;  END;
         IF R6 ¬= 0 THEN MEMORY;
         NORMREAD;
CA16:    R2 := OPERAND XOR _1;  B5 := R2;
         R8 := 15;
      END;
      BEGIN                                              COMMENT OPCODE = 17;
         IF CHARADDR THEN BEGIN  MEMORY;  GOTO CA17;  END;
         IF R6 ¬= 0 THEN MEMORY;
         NORMREAD;
CA17:    IF OPERAND < 0 THEN OPERAND := OPERAND XOR _1;
         B5 := OPERAND;
         R8 := 15;
      END;
      BEGIN                                              COMMENT OPCODE = 20;
         IF R7 = 0 THEN GOTO TIME20;
         R5 := CPUBASE + INDEXG   + R7;    COMMENT R5 = B REG ADDRESS;
         IF CHARADDR THEN BEGIN  MEMORY;  GOTO CA20;  END;
         IF R6 ¬= 0 THEN MEMORY;
         NORMREAD;
CA20:    R4 := B5 AND #E0000;
         R3 := OPERAND AND #FFFF;
         R4 := R4 OR R3;       B5 := R4;
TIME20:R8 := 20;
      END;
      BEGIN                                              COMMENT OPCODE = 21;
         IF R7 = 0 THEN GOTO TIME21;
         R5 := CPUBASE + INDEXG   + R7;    COMMENT R5 = B REG ADDRESS;
         IF CHARADDR THEN BEGIN  MEMORY;  GOTO CA21;  END;
         IF R6 ¬= 0 THEN MEMORY;  NORMREAD;
CA21:    R2 := B5;  R1 := R2 AND #E0000;  R2 := R2 AND #FFFF;
         R3 := OPERAND;  ADD;  R6 := R6 AND #FFFF OR R1;  B5 := R6;
TIME21: R8 := 20;
      END;
      BEGIN                                              COMMENT OPCODE = 22;
         IF R7 = 0 THEN GOTO TIME22;
         R5 := CPUBASE + INDEXG   + R7;    COMMENT R5 = B REG ADDRESS;
         IF CHARADDR THEN BEGIN  MEMORY;  GOTO CA22;  END;
         IF R6 ¬= 0 THEN MEMORY;  NORMREAD;
CA22:    R2 := B5;  R1 := R2 AND #E0000;  R2 := R2 AND #FFFF;
         R3 := OPERAND XOR _1;  ADD;  R6 := R6 AND #FFFF OR R1;  B5 := R6;
```

```
TIME22: R8 := 20;
    END;
    BEGIN                                                          COMMENT OPCODE = 23;
      IF R7 = 0 THEN BEGIN
        R5 := R5 - R5;   GOTO OP23;   END;
      R3 := CPUBASE + R7 + INDEXG;  R5 := B3 AND #FFFF;
OP23: IF CHARADDR THEN
      BEGIN  R2 := R5;   R1 := 2;   MEMORY;   END
      ELSE  NORMSTORE;
      R8 := 15;
    END;
    BEGIN                                                          COMMENT OPCODE = 24;
      R5 := B5;
      IF CHARADDR THEN
      BEGIN  R2 := R5;   R1 := 2;   MEMORY;   END
      ELSE  NORMSTORE;
      R8 := 15;
    END;
    BEGIN                                                          COMMENT OPCODE = 25;
      IF ¬ INDEX THEN GOTO TIME25;
      R5 := B5;
      IF CHARADDR THEN
      BEGIN  R2 := R5;   R1 := 2;   MEMORY;   END
      ELSE  NORMSTORE;
      R5 := INDEXREGLOC;  R2 := B5 AND #FFFF;  R3 := 1;  ADD;
      R6 := R6 AND #FFFF;  R2 := R2 AND #F0000 OR R6;    B5 := R2;
TIME25: R8 := 15;
    END;
    BEGIN                                                          COMMENT OPCODE = 26;
      R5 := B5 XOR _1;
      IF CHARADDR THEN
      BEGIN  R2 := R5;   R1 := 2;   MEMORY;   END
      ELSE  NORMSTORE;
      R8 := 15;
    END;
    BEGIN                                                          COMMENT OPCODE = 27;
      R5 := B5;
      IF R5 < 0 THEN R5 := R5 XOR _1;
      IF CHARADDR THEN
      BEGIN  R2 := R5;   R1 := 2;   MEMORY;   END
      ELSE  NORMSTORE;
      R8 := 15;
    END;
    GOTO ILLEGALINST;                                              COMMENT OPCODE = 30;
    GOTO ILLEGALINST;                                              COMMENT OPCODE = 31;
    BEGIN                                                          COMMENT OPCODE = 32;
      IF CHARADDR THEN GOTO ILLEGALINST;
      IF R7 > 12 THEN GOTO TIME32;
```

```
      R7 := R7 SHLL 1 OR R6;  R4 := OPERAND;
      MEMORY;  R1 := R1 SHLL R7 XOR _1;
      IF INST ¬= 1 THEN  R2 := OPERAND AND R1
      ELSE BEGIN R1 := R1 XOR _1; R2 := OPERAND OR R1; END;
      OPERAND := R4;  R1 := 2;  MEMORY;
TIME32:R8 := 25;
    END;
    BEGIN                                            COMMENT OPCODE = 33;
      INST := R1;  OPCODE := 26;  GOTO CASESTATE;
    END;
    BEGIN                                            COMMENT OPCODE = 34;
      IF CHARADDR THEN GOTO CA34;
      IF R6 = 0 THEN GOTO TIME34;
CA34: R3 := B5;  COMMENT R3 = (A);
      IF INST = R1 THEN R3 := R3 XOR _1;
      IF REPLACE THEN
        R7 := REPLACESTOREVAL
      ELSE
        R7 := OPERAND;
      R4 := R6;  MEMORY;  IF ¬CHARADDR THEN NORMREAD;
      R2 := OPERAND;  ADD;  R5 := R5 + ANEXT;  B5 := R6;
      R5 := R6;  R6 := R4;  OPERAND := R7;
      IF CHARADDR THEN
      BEGIN  R2 := R5;  R1 := 2;  MEMORY;  END
      ELSE  NORMSTORE;
TIME34: R8 := 25;
    END;
    BEGIN                                            COMMENT OPCODE = 35;
      IF CHARADDR THEN GOTO CA35;
      IF R6 = 0 THEN GOTO TIME35;
CA35: IF REPLACE THEN
        R7 := REPLACESTOREVAL
      ELSE
        R7 := OPERAND;
      R4 := R6;  MEMORY;  IF ¬CHARADDR THEN NORMREAD;  R2 := OPERAND;
      IF INST = R1 THEN R3 := _1 ELSE R3 := R1;
      ADD;  B5 := R6;  R5 := R6;  R6 := R4;  OPERAND := R7;
      IF CHARADDR THEN
      BEGIN  R2 := R5;  R1 := 2;  MEMORY;  END
      ELSE  NORMSTORE;
TIME35: R8 := 25;
    END;
    BEGIN                                            COMMENT OPCODE = 36;
      INST := R1;  OPCODE := 28;  GOTO CASESTATE;
    END;
    BEGIN                                            COMMENT OPCODE = 37;
      INST := R1;  OPCODE := 29;  GOTO CASESTATE;
    END;
```

```
    BEGIN                                               COMMENT OPCODE =40;
      IF CHARADDR THEN BEGIN  MEMORY;  GOTO CA40;  END;
      IF R6 ¬= 0 THEN  MEMORY;  NORMREAD;
CA40: R4 := R4 - R4;  R3 := B5;
      IF R3 < 0 THEN BEGIN R3 := R3 XOR _1;  R4 := R4 + R1;  END;
      IF OPERAND < 0 THEN
      BEGIN
        OPERAND := OPERAND XOR _1;
        R4 := R4 - R1;
      END;
      R3 := R3 * OPERAND;
      IF R4 ¬= 0 THEN BEGIN R2 := R2 XOR _1;  R3 := R3 XOR _1; END;
      B5 := R3;  R5 := R5 + ANEXT;  B5 := R2;
      R8 := 75;
    END;
    BEGIN                                               COMMENT OPCODE = 41;
      IF CHARADDR THEN BEGIN  MEMORY;  GOTO CA41;  END;
      IF R6 ¬= 0 THEN  MEMORY;  NORMREAD;
CA41: R3 := B5;  R5 := R5 + ANEXT;
      R2 := B5;                          COMMENT R2, R3 = A(A+1), A(A);
      R4 := R4 - R4;  R1 := R4;  R6 := R4;
      IF R2 < 0 THEN
      BEGIN                    COMMENT MAKE DIVIDEND POSITIVE;
        R3 := R3 XOR _1;  R2 := R2 XOR _1;  R1 := R1 + 1;  R6 := R1;
      END;                   COMMENT R6 INDICATES NEGATIVE REMAINDER;
      IF OPERAND < 0 THEN
      BEGIN
        OPERAND := OPERAND XOR _1;  R1 := R1 - 1;
      END;                  COMMENT IF R1 = 0 THEN QUOTIENT IS +;
      IF OPERAND = 0 THEN
      BEGIN
        ASR := ASR OR #08;  GOTO TIME41;
      END;
      SLDL(R2,1);   COMMENT PREVENT ANY POSSIBLE OC9;
      CLR(OPERAND,R2);
      IF <= THEN BEGIN ASR := ASR OR #08;  GOTO TIME41;  END;
      SRDL(R2,1);
      R3 := R3 / OPERAND;    COMMENT R2 = REMAINDER, R3 = QUOTIENT;
      IF R1 ¬= 0 THEN R3 := R3 XOR _1;
      IF R6 ¬= 0 THEN R2 := R2 XOR _1;
      B5 := R2;  R5 := R5 - ANEXT;  B5 := R3;
TIME41: R8 := 145;
    END;
    BEGIN                                               COMMENT OPCODE = 42;
      IF CHARADDR THEN GOTO ILLEGALINST;
      IF R7 > 3 THEN GOTO TIME42;
      R7 := R7 SHLL 3 OR R6;
      MEMORY;  R2 := R1 SHLL R7 AND OPERAND;
```

```
      IF R2 = 0 THEN ASR := ASR OR #06
       ELSE ASR := ASR AND #FFFFFFF7;
TIME42: R8 := 15;
    END;
    BEGIN                                            COMMENT OPCODE = 43;
      IF R7 = 0 THEN GOTO TIME43;
      IF CHARADDR THEN  BEGIN  MEMORY;  GOTO CA43;  END;
      IF R6 ¬= 0 THEN  MEMORY;  NORMREAD;
CA43: R5 := CPUBASE + R7 + INDEXG;  R4 := B5;  R2 := R4 AND #FFFF;
      IF R2 >= OPERAND THEN
      BEGIN  R4 := R4 AND #E0000;  B5 := R4;
        ASR := ASR OR #01;  END
      ELSE BEGIN  R4 := R4 AND #EFFFF + 1 AND #EFFFF;
        B5 := R4;  ASR := ASR AND #FFFFFFFE;
      END;
TIME43: R8 := 20;
    END;
    BEGIN                                            COMMENT OPCODE = 44;
      IF CHARADDR THEN  BEGIN  MEMORY;  GOTO CA44;  END;
      IF R6 ¬= 0 THEN  MEMORY;  NORMREAD;
CA44: R5 := B5;  ASR := ASR AND #FFFFFFF9;
      IF R5 = OPERAND THEN BEGIN ASR := ASR OR #06;  GOTO TIME44;  END;
      IF R5 > OPERAND THEN ASR := ASR OR #02;
TIME44: R8 := 15;
    END;
    BEGIN                                            COMMENT OPCODE = 45;
      IF CHARADDR THEN  BEGIN  MEMORY;  GOTO CA45;  END;
      IF R6 ¬= 0 THEN  MEMORY;  NORMREAD;
CA45: R4 := B5;  R5 := R5 + ANEXT;  R5 := B5;
      IF R5 > OPERAND AND OPERAND >= R4 THEN ASR := ASR AND #FFFFFFFE
      ELSE ASR := ASR OR #1;
      R8 := 15;
    END;
    BEGIN                                            COMMENT OPCODE = 46;
      IF CHARADDR THEN  BEGIN  MEMORY;  GOTO CA46;  END;
      IF R6 ¬= 0 THEN  MEMORY;  NORMREAD;
CA46: R4 := B5;  R5 := R5 + ANEXT;  R5 := B5;
      ASR := ASR AND #FFFFFFF9;  R4 := R4 AND OPERAND;
      IF R5 > R4 THEN BEGIN  ASR := ASR OR #06;  GOTO TIME46;  END;
      IF R5 = R4 THEN ASR := ASR OR #02;
TIME46: R8 := 15;
    END;
    BEGIN                                            COMMENT OPCODE = 47;
      IF CHARADDR THEN  BEGIN  MEMORY;  GOTO CA47;  END;
      IF R6 ¬= 0 THEN  MEMORY;  NORMREAD;
CA47: R8 := ASR;  R2 := OPERAND;  R3 := B5;  R5 := R5 + ANEXT;
      R5 := B5;  ADD;  IF R6 < 0 THEN R6 := R6 XOR _1;
      ASR := R8 AND #FFFFFFF9;
```

```
     IF R6 > R5 THEN BEGIN ASR := ASR OR #06;  GOTO TIME47;  END;
     IF R6 = R5 THEN ASR := ASR OR #02;
TIME47: R8 := 15;
   END;
  END;                                  COMMENT END CASE STATEMENT;
ENDCASE: TIME;  MEM(CPUBASE + 100) := ASR;  GOTO FINISH;
ILLEGALINST:R1 := 2;  R2 := R1;  INTERRUPT;
PRIVINST:R1 := 2;  R2 := 3;  INTERRUPT;
FINISH:R14 := SAVE14;
END;          COMMENT END PROCEDURE FULLWORD2;
```

```
SEGMENT PROCEDURE FULLWORD3 (R14);
BEGIN
 INTEGER SAVE14;
 SAVE14 := R14;
 OPCODE := OPCODE - 39;
 CASE OPCODE OF
 BEGIN                          COMMENT START OF CASE STATEMENT;
  BEGIN                                      COMMENT OPCODE = 50X;
   R3 := R6 AND #01;  IF R3 > 0 THEN GOTO ILLEGALINST;
   R6 := R6 SHRL 1 + R1;
   R4 := INST AND #E000 SHLL 4;
   R7 := INST AND #1FFF + INDEXREGVAL OR R4;
   R4 := B5;  R5 := R5 + ANEXT;  R5 := B5;
OP500:CASE R6 OF
   BEGIN                         COMMENT CASE OPCODE OF 500 - 503;
    BEGIN                                    COMMENT OPCODE = 500;
     R5 := R5 AND R4;  R2 := R2 - R2;  R4 := R2;
     FOR R3 := 1 STEP 1 UNTIL 31 DO
     BEGIN IF R5 < R4 THEN R2 := R2 + R1; R5 := R5 SHLL 1; END;
     SRDL(R2,1);
     IF INST = R1 THEN
     BEGIN  IF R3 < 0 THEN GOTO DO500 ELSE GOTO TIME50X; END;
     IF R3 < 0 THEN GOTO TIME50X;
DO500:    MEM(CPUBASE + 328) := R7;
    END;
    BEGIN                                    COMMENT OPCODE = 501;
     INST := R1;  R6 := R1;  GOTO OP500;
    END;
    BEGIN                                    COMMENT OPCODE = 502;
     IF R4 ¬= 0 AND R5 ¬= 0 THEN GOTO TIME50X;
     MEM(CPUBASE + 328) := R7;
    END;
    BEGIN                                    COMMENT OPCODE = 503;
     IF R4 = 0 AND R5 = 0 THEN GOTO TIME50X;
     MEM(CPUBASE + 328) := R7;
    END;
   END;                          COMMENT END CASE OPCODE 500 - 503;
TIME50X: R8 := 20;
  END;
  BEGIN                                      COMMENT OPCODE = 51X;
   R3 := R6 AND #01;  IF R3 > 0 THEN GOTO ILLEGALINST;
   R6 := R6 SHRL 1 + R1;
   R4 := INST AND #E000 SHLL 4;
   R7 := INST AND #1FFF + INDEXREGVAL OR R4;  R5 := B5;
   CASE R6 OF
   BEGIN                         COMMENT CASE OPCODE OF 510 - 513;
    BEGIN                                    COMMENT OPCODE = 510;
     IF R5 <  0 THEN GOTO TIME51X;
```

```
          MEM(CPUBASE + 328) := R7;
        END;
        BEGIN                                      COMMENT OPCODE = 511;
          IF R5  > 0 THEN GOTO TIME51X;
          MEM(CPUBASE + 328) := R7;
        END;
        BEGIN                                      COMMENT OPCODE = 512;
          IF R5 ¬= ) THEN GOTO TIME51X;
          MEM(CPUBASE + 328) := R7;
        END;
        BEGIN                                      COMMENT OPCODE = 513;
          IF R5  = 0 THEN GOTO TIME51X;
          MEM(CPUBASE + 328) := R7;
        END;
      END;                             COMMENT END CASE OPCODE 510 - 513;
TIME51X: R8 := 15;
    END;
    BEGIN                                          COMMENT OPCODE = 52X;
      R3 := R6 AND #01;  IF R3 > 0 THEN GOTO ILLEGALINST;
      R6 := R6 SHRL 1 + R1;
      CASE R6 OF
      BEGIN                             COMMENT CASE OPCODE OF 520 - 523;
        BEGIN                                      COMMENT OPCODE = 520;
          IF R7 = 0 THEN GOTO TIME520;
          R5 := CPUBASE + INDEXG + R7;  R6 := MEM(CPUBASE + 328);
          B5 := R6;  R4 := INST AND #E000 SHLL 4;
          R7 := INST AND #1FFF + INDEXREGVAL OR R4;
          MEM(CPUBASE + 328) := R7;
TIME520: R8 := 18;
        END;
        BEGIN                                      COMMENT OPCODE = 521;
          IF R7 = 0 THEN GOTO TIME521;
          R5 := CPJBASE + INDEXG + R7;  R4 := B5;
          IF R4 <= 0 THEN GOTO TIME521;
          R4 := R4 - R1;  B5 := R4;
          R4 := INST AND #E000 SHLL 4;
          R7 := INST AND #1FFF + INDEXREGVAL OR R4;
          MEM(CPUBASE + 328) := R7;
TIME521: R8 := 18;
        END;
        BEGIN                                      COMMENT OPCODE = 522;
          R3 := INDEXREGLOC;  R3 := B3;
          R4 := INST AND #FFFF + R3 AND #FFFF;
          R2 := R3 AND #E0000 OR R4;
          MEM(CPUBASE + 328) := R2;
          R8 := 15;
        END;
        BEGIN                                      COMMENT OPCODE = 523;
```

```
      R4 := INST AND #E000 SHLL 4;
      R7 := INST AND #1FFF + INDEXREGVAL OR R4 + R1;
      MEM(CPUBASE + 328) := R7;  ASR := ASR AND #FFFF7FFF;
      MEMORY;  INST := OPERAND;  HALFWORD;
      R8 := 15;
    END;
  END;                             COMMENT END CASE OPCODE 520 - 523;
END;
BEGIN                                           COMMENT OPCODE = 53X;
  R3 := R6 AND #01;  IF R3 > 0 THEN GOTO ILLEGALINST;
  R5 := R6 SHRL 1;  IF R5 >= 2 THEN GOTO OP532;
  R7 := R7 SHRL 2;  R6 := R6 SHRL 1 SHLL 3 OR R7 + R1;
  CASE R6 OF
  BEGIN                           COMMENT CASE OPCODE OF 5300 - 5317;
    BEGIN                                      COMMENT OPCODE = 5300;
      R4 := ASR AND #08;
      IF R4 > 0 THEN BEGIN ASR := ASR XOR R4; GOTO TIME53X; END;
      GOTO  SETBR53;
    END;
    BEGIN                                      COMMENT OPCODE = 5301;
      R4 := ASR AND #08;
      IF R4 = 0 THEN GOTO TIME53X;
      ASR := ASR XOR R4;  GOTO SETBR53;
    END;
    GOTO ILLEGALINST;
    GOTO ILLEGALINST;
    GOTO ILLEGALINST;
    GOTO ILLEGALINST;
    GOTO ILLEGALINST;
    GOTO ILLEGALINST;
    BEGIN                                      COMMENT OPCODE = 5310;
      R4 := ASR AND #04;
      IF R4 > 0 THEN GOTO TIME53X ELSE GOTO SETBR53;
    END;
    BEGIN                                      COMMENT OPCODE = 5311;
      R4 := ASR AND #04;
      IF R4 = 0 THEN GOTO TIME53X ELSE GOTO SETBR53;
    END;
    BEGIN                                      COMMENT OPCODE = 5312;
      R4 := ASR AND #06;
      IF R4 = 2 THEN GOTO SETBR53 ELSE GOTO TIME53X;
    END;
    BEGIN                                      COMMENT OPCODE = 5313;
      R4 := ASR AND #02;
      IF R4 > 0 THEN GOTO SETBR53 ELSE GOTO TIME53X;
    END;
    BEGIN                                      COMMENT OPCODE = 5314;
      R4 := ASR AND #02;
```

```
          IF R4 = 0 THEN GOTO SETBR53 ELSE GOTO TIME53X;
        END;
        BEGIN                                     COMMENT OPCODE = 5315;
          R4 := ASR AND #06;
          IF R4 ¬= 2 THEN GOTO SETBR53 ELSE GOTO TIME53X;
        END;
        BEGIN                                     COMMENT OPCODE = 5316;
          R4 := ASR AND #01;
          IF R4 > 0 THEN GOTO SETBR53 ELSE GOTO TIME53X;
        END;
        BEGIN                                     COMMENT OPCODE = 5317;
          R4 := ASR AND #01;
          IF R4 = 0 THEN GOTO SETBR53 ELSE GOTO TIME53X;
        END;
      END;                       COMMENT  END CASE OPCODE OF 5300 - 5317;
SETBR53:R4 := INST AND #E000 SHLL 4;
      R7 := INST AND #1FFF + INDEXREGVAL OR R4;
      MEM(CPUBASE + 328) := R7;  GOTO TIME53X;
OP532:R5 := R5 - R1;
      R7 := R7 + 1;  R3 := CPUBASE + 336;  R3 := B3;
      CASE R7 OF
      BEGIN                        COMMENT CASE OF SPECIAL A DESIGNATOR;
        GOTO JUMPOP53;                                    COMMENT A = 0;
        BEGIN                                             COMMENT A = 1;
          R3 := R3 AND #F0000000;
          IF R3 > 0 THEN GOTO JUMPOP53;
        END;
        BEGIN                                             COMMENT A = 2;
          R3 := R3 AND #F000000;
          IF R3 > 0 THEN GOTO JUMPOP53;
        END;
        BEGIN                                             COMMENT A = 3;
          R3 := R3 AND #F00000;
          IF R3 > 0 THEN GOTO JUMPOP53;
        END;
        BEGIN                                             COMMENT A = 4;
          IF ¬ EXECSTATE THEN GOTO PRIVINST ELSE GOTO ENDIT;
        END;
        BEGIN                                             COMMENT A = 5;
          IF ¬ EXECSTATE THEN GOTO PRIVINST;
          R3 := R3 AND #F000;
          IF R3 > 0 THEN GOTO ENDIT ELSE GOTO JUMPOP53;
        END;
        BEGIN                                             COMMENT A = 6;
          IF ¬ EXECSTATE THEN GOTO PRIVINST;
          R3 := R3 AND #F00;
          IF R3 > 0 THEN GOTO ENDIT ELSE GOTO JUMPOP53;
        END;
```

```
        BEGIN                                             COMMENT A = 7;
          IF ¬ EXECSTATE THEN GOTO PRIVINST;
          R3 := R3 AND #F0;
          IF R3 > 0 THEN GOTO ENDIT ELSE GOTO JUMPOP53;
        END;
      END;                     COMMENT END CASE OF SPECIAL A DESIGNATOR;
      GOTO TIME53X;
ENDIT: R3 := ASR SHRL 20 SHLL 1;   R14 := 1;  CPUSTATE(R3) := R14;
      MEM(CPUBASE + 100) := ASR;  R8 := 15;  TIME;
      R1 := 2;  BRANCH;
JUMPOP53:CASE R5 OF
      BEGIN                            COMMENT CASE OPCODE OF 532 - 533;
        BEGIN                                      COMMENT OPCODE = 532;
          R1 := 2;  R2 := MEM(CPUBASE + 328);  MEMORY;
          R4 := INST AND #E000 SHLL 4;
          R7 := INST + 1 AND #1FFF + INDEXREGVAL OR R4;
          MEM(CPUBASE + 328) := R7;
          R8 := 30;  GOTO ENDCASE;
        END;
        BEGIN                                      COMMENT OPCODE = 533;
          R4 := INST AND #E000 SHLL 4;
          R7 := INST AND #1FFF + INDEXREGVAL OR R4;
          MEM(CPUBASE + 328) := R7;
          R8 := 30;  GOTO ENDCASE;
        END;
      END;                        COMMENT END CASE OPCODE OF 532 - 533;
TIME53X: R8 := 15;
    END;
    BEGIN                                           COMMENT OPCODE = 54;
      IF REPEAT AND ¬EXECSTATE THEN GOTO PRIVINST;
      R7 := R7 SHLL 1 OR R6;
      IF ¬EXECSTATE AND R7 > 15 THEN GOTO PRIVINST;
OP54: IF R7 > 63 THEN GOTO ILLEGALINST;  R4 := R4 - R4;
      IC(R4,CMRADDR(R7));  IF R4 = 255 THEN GOTO ILLEGALINST;
      R3 := R4;  R4 := R4 SHLL 2 +CPUBASE;  R1 := 1;  MEMORY;
      IF R3 = 25 THEN
      BEGIN
        R3 := B4 AND #7F8000;  OPERAND := OPERAND AND #7FFF;
        R3 := R3 OR OPERAND;  B4 := R3;
      END ELSE
        B4 := OPERAND;
      R8 := 15;  TIME;
      IF REPEAT THEN
      BEGIN
        RESET(REPEATCMR);
        R4 := REPEATCOUNT;
        R4 := R4 - 1;  REPEATCOUNT := R4;
        IF R4 <= 0 THEN
```

```
        BEGIN
          R3 := B7ADDR;  R8 := 0;  B3 := R8;
          RESET(REPEAT);  GOTO ENDCASE;
        END:
        REPEATTERM;
        IF ¬REPEAT THEN
        BEGIN
          R3 := B7ADDR;  B3 := R4;  R8 := 0;
          GOTO ENDCASE;
        END;
        IF INDEX THEN
        BEGIN
          R2 := INDEXREGLOC;  R3 := B2 + REPEATINDEX AND #FFFF;
          B2 := R3;  OPERAND := R3;
        END ELSE
          OPERAND := OPERAND - OPERAND;
        R2 := INST AND #1FFF;  OPERAND := OPERAND + R2;
        R2 := BASEREGLOC;  R2 := B2 AND #3FFFF;
        OPERAND := OPERAND + R2;  R7 := R7 + 1;
        GOTO OP54;
      END:
      R8 := 0;
    END:
    BEGIN                                           COMMENT OPCODE = 55;
      R7 := R7 SHLL 1 OR R6 + 64;
      IF ¬EXECSTATE THEN GOTO PRIVINST;
OP55: IF R7 > 127 THEN GOTO ILLEGALINST;
      R5 := R5 - R5;  IC(R5,CMRADDR(R7));
      IF R5 = 255 THEN GOTO ILLEGALINST;
      R5 := R5 SHLL 2 + CPUBASE;  R1 := 1;  MEMORY;  B5 := OPERAND;
      R8 := 15;  TIME;
      IF REPEAT THEN
      BEGIN
        RESET(REPEATCMR);
        R4 := REPEATCOUNT;
        R4 := R4 - 1;  REPEATCOUNT := R4;
        IF R4 <= 0 THEN
        BEGIN
          R3 := B7ADDR;  R8 := 0;  B3 := R8;
          RESET(REPEAT);  GOTO ENDCASE;
        END:
        REPEATTERM;
        IF ¬REPEAT THEN
        BEGIN
          R3 := B7ADDR;  B3 := R4;  R8 := 0;
          GOTO ENDCASE;
        END;
        IF INDEX THEN
```

```
        BEGIN
          R2 := INDEXREGLOC;  R3 := B2 + REPEATINDEX AND #FFFF;
          B2 := R3;  OPERAND := R3;
        END ELSE
          OPERAND := OPERAND - OPERAND;
        R2 := INST AND #1FFF;  OPERAND := OPERAND + R2;
        R2 := BASEREGLOC;  R2 := B2 AND #3FFFF;
        OPERAND := OPERAND + R2;  R7 := R7 + 1;
        GOTO OP55;
      END;
      R8 := 0;
    END;
    BEGIN                                                    COMMENT OPCODE = 56;
      IF REPEAT AND ¬EXECSTATE THEN GOTO PRIVINST;
      R7 := R7 SHLL 1 OR R6;
      IF ¬EXECSTATE AND R7 > 15 THEN GOTO PRIVINST;
OP56: IF R7 > 63 THEN GOTO ILLEGALINST;  R4 := R4 - R4;
      IC(R4,CMRADDR(R7));  IF R4 = 255 THEN GOTO ILLEGALINST;
      R4 := R4 SHLL 2 + CPUBASE;  R2 := B4;  R1 := 2;  MEMORY;
      R8 := 15;  TIME;
      IF REPEAT THEN
      BEGIN
        RESET(REPEATCMR);
        R4 := REPEATCOUNT;
        R4 := R4 - 1;  REPEATCOUNT := R4;
        IF R4 <= 0 THEN
        BEGIN
          R3 := B7ADDR;  R8 := 0;  B3 := R8;
          RESET(REPEAT);  GOTO ENDCASE;
        END;
        REPEATTERM;
        IF ¬REPEAT THEN
        BEGIN
          R3 := B7ADDR;  B3 := R4;  R8 := 0;
          GOTO ENDCASE;
        END;
        IF INDEX THEN
        BEGIN
          R2 := INDEXREGLOC;  R3 := B2 + REPEATINDEX AND #FFFF;
          B2 := R3;  OPERAND := R3;
        END ELSE
          OPERAND := OPERAND - OPERAND;
        R2 := INST AND #1FFF;  OPERAND := OPERAND + R2;
        R2 := BASEREGLOC;  R2 := B2 AND #3FFFF;
        OPERAND := OPERAND + R2;  R7 := R7 + 1;
        GOTO OP56;
      END;
      R8 := 0;
```

```
    END;
    BEGIN                                             COMMENT OPCODE = 57;
      R7 := R7 SHLL 1 OR R6 + 64;
      IF ¬EXECSTATE THEN GOTO PRIVINST;
OP57: IF R7 > 127 THEN GOTO ILLEGALINST;
      R5 := R5 - R5;  IC(R5,CMRADDR(R7));
      IF R5 = 255 THEN GOTO ILLEGALINST;
      R5 := R5 SHLL 2 + CPUBASE;  R2 := B5;  R1 := 2;  MEMORY;
      R8 := 15;  TIME;
      IF REPEAT THEN
      BEGIN
        RESET(REPEATCMR);
        R4 := REPEATCOUNT;
        R4 := R4 - 1;  REPEATCOUNT := R4;
        IF R4 <= 0 THEN
        BEGIN
          R3 := B7ADDR;  R8 := 0;  B3 := R8;
          RESET(REPEAT);  GOTO ENDCASE;
        END;
        REPEATTERM;
        IF ¬REPEAT THEN
        BEGIN
          R3 := B7ADDR;  B3 := R4;  R8 := 0;
          GOTO ENDCASE;
        END;
        IF INDEX THEN
        BEGIN
          R2 := INDEXREGLOC;  R3 := B2 + REPEATINDEX AND #FFFF;
          B2 := R3;  OPERAND := R3;
        END ELSE
          OPERAND := OPERAND - OPERAND;
        R2 := INST AND #1FFF;  OPERAND := OPERAND + R2;
        R2 := BASEREGLOC;  R2 := B2 AND #3FFFF;
        OPERAND := OPERAND + R2;  R7 := R7 + 1;
        GOTO OP57;
      END;
      R8 := 0;
    END;
  END;                                         COMMENT END CASE STATEMENT;
ENDCASE:  TIME;  MEM(CPUBASE + 100) := ASR;  GOTO FINISH;
ILLEGALINST:R1 := 2;  R2 := R1;  INTERRUPT;
PRIVINST:R1 := 2;  R2 := 3;  INTERRUPT;
FINISH:R14 := SAVE14;
END;                                       COMMENT END PROCEDURE FULLWORD3;
```

```
INITIALIZE;
 ASR := CPU0(780);
SETUP:  R5 := 4;
 R6 := R6 - R6;  IC(R6,NCPU);  R6 := R6 - 1 SHLL 1;
 R7 := ASR SHRL 20 + 1 SHLL 1;  IF R7 >R6 THEN R7 := 0;
 MVC(6,REPEAT,ZERO);   COMMENT RESET CPU STATE FLAG TO FALSE;
 R1 := REALCLOCK;  R1 := R1 + 20;
 REALCLOCK := R1;   COMMENT UPDATE REAL CLOCK;
 R1 := IMONCLOCK;
 IF R1 >= 0 THEN
 BEGIN
  R1 := R1 - 20;  IMONCLOCK := R1;   COMMENT DECREMENT IOC MON CLOCK;
  IF R1 < 0 THEN      CCMMENT CHECK FOR IOC MON CLOCK < 0;
  BEGIN
   FOR R3 := 0 STEP 2 UNTIL R6 DO
   BEGIN
    R4 := CPUSTATE(R3);
    IF R4 = 2 OR R4 = 4 THEN
    BEGIN      COMMENT SET UP FOR INTERRUPT;
     R4 := 5;  CPUSTATE(R3) := R4;  GOTO COMLOOP2;
    END;
   END;
  END;
 END;
COMLOOP2: FOR R3 := R7 STEP 2 UNTIL R6 DO
 BEGIN
  R8 := R3 SHLL 1;
  R4 := CPUSTATE(R3);
  CPUBASE := CPUCM(R8);
  ASR := MEM(CPUBASE + 100);
  CASE R4 OF
  BEGIN
   NULL;   COMMENT CASE 1 HALT;
   NULL;      COMMENT CASE 2 WAIT;
   BEGIN      COMMENT CASE 3 INTERPROCESSOR INTERRUPT;
    CPUSTATE(R3) := R5;
    R1 := 2;  R2 := 0;
    INTERRUPT;
   END;
   BEGIN      COMMENT CASE 4 ACTIVE;
    GOTO EXEC;
   END;
   BEGIN      COMMENT CASE 5 IOC MON CLOCK INTERRUPT;
    CPUSTATE(R3) := R5;
    R1 := 3;  R2 := 10;  INTERRUPT;
   END;
  END;        COMMENT END CASE ON CPU STATE;
 END;
```

```
  IF R7 > 0 THEN GOTO SETUP;
  GOTO BOMB;
EXEC:R7 := ASR AND #F0000;
IF R7 = 0 THEN RESET(EXECSTATE) ELSE SET(EXECSTATE);
IF ¬REPEAT THEN
BEGIN  COMMENT REPEAT CONDITION DOES NOT APPLY, FETCH NEXT INSTRUCTION;
  R1 := 0;  MEMORY;
END ELSE
BEGIN  COMMENT REPEAT NEXT INSTRUCTION;
  IF REPEATCMR THEN GOTO SKIPREPEAT;
  R1 := B7ADDR;  R2 := B1 - 1;
  IF R2 < 0 THEN
  BEGIN
    RESET(REPEAT);  RESET(REPLACE);  RESET(COMPARE);  GOTO SETUP;
  END;
  R3 := REPEATSKIP - 1;  REPEATSKIP := R3;
  IF R3 <= 0 THEN
  BEGIN
    REPEATTERM;  IF ¬REPEAT THEN GOTO SETUP;
    B1 := R2;
    IF INDEX THEN
    BEGIN
      R2 := INDEXREGLOC;  R3 := B2 + REPEATINDEX;  B2 := R3;
    END;
  END ELSE
    B1 := R2;
SKIPREPEAT:END;
OPCODE := INST SHRL 26; CCMMENT GETTING THE OP CODE;
IF OPCODE > #30 THEN
  BEGIN  HALFWORD;  GOTO SETUP;  END;
EXECREMOTE:R7 := ASR SHLL 21;  COMMENT ? TASK OR INT ACCUM/INDEX REGS;
IF R7 >= 0 THEN
BEGIN
  R6 := 0;  ACCG := R6;  R6 := 32;  INDEXG := R6;
END ELSE
BEGIN
  R6 := 104;  ACCG := R6;  R6 := 136;  INDEXG := R6;
END;
R7 := ASR SHLL 20;  COMMENT ? TASK OR INT BASE REGS;
IF R7 >= 0 THEN
BEGIN
  R6 := 64;  BASEG := R6;
END ELSE
BEGIN
  R6 := 168;  BASEG := R6;
END;
COMPUTEY:  RESET(CHARADDR);  RESET(ICIR);
INDIRECT:R7 := INST AND #10000;
```

```
IF R7 > 0 THEN
BEGIN  COMMENT INDIRECT ADDRESSING;
  IAWADDR := OPERAND;  R1 := 6;  MEMORY;
  IAW := OPERAND;  R6 := OPERAND AND #FFFFF;
  INST := INST AND #FFF00000 OR R6;  OPERAND := OPERAND SHRL 30;
  IF OPERAND = 0 THEN  COMPUTESPECY ELSE COMPUTEY;
  SET(IDIR);  GOTO INDIRECT;
END ELSE
BEGIN
  IF ¬IDIR THEN GOTO FINALY;  COMMENT NO INDIRECT ADDRESSING;
  R7 := IAW SHRL 30;
  IF R7 = 0 OR R7 = 2 THEN GOTO FINALY;
  IF OPCODE > 39 THEN
  BEGIN  COMMENT ILLEGAL TO CHARACTER ADDRESS FOR OPCODE > 47;
    R1 := 2;  R2 := R1;  INTERRUPT;
  END;  SET(CHARADDR);
  COMMENT WHEN HERE CHARACTER ADDRESSING ALLOWED;
  R6 := IAW;  R5 := R6 SHRL 20 AND #3FF;  R4 := R5 AND #1F;
  IAWP := R4;  R5 := R5 SHRL 5;  IAWW := R5;
  IF R5 < 1 THEN
  BEGIN  R1 := 2;  R2 := R1;  INTERRUPT;  END;
  IF R7 = 1 THEN GOTO FINALY;
  COMMENT UPDATING IAW FOR SEQUENTIAL CHARACTER ADDRESSING.  THE IAW IS
  CHANGED ACCORDING TO P & W FORMULAS AND IS STORED AWAY.  IF IN THE
  REPEAT MODE AND SEQUENTIAL CHARACTER ADDRESSING IS SPECIFIED, THEN
  SINGLE CHARACTER ADDRESSING IS DONE, THE IAW IS MODIFIED, AND
  STORED AWAY,  AND B7 IS SET TO ZERO, THEREBY TERMINATING THE REPEAT
  MODE.  THIS IS DUE TO AN ABNORMALITY IN THE UYK - 7 AND IS NOT
  DOCUMENTED;
  R3 := R4 - R5;
  IF R3 >= 0 THEN
    R4 := R3
  ELSE
  BEGIN
    R3 := R6 AND #1FFF;  R3 := R3 + 1 AND #1FFF;
    R6 := R6 AND #FFFFE000 OR R3;  R4 := 32 - R5;
  END;
  R4 := R4 SHLL 20;  R5 := R5 SHLL 25 OR R4;
  R6 := R6 AND #C00FFFFF OR R5;
  IF REPEAT THEN
  BEGIN
    R3 := B7ADDR;  R2 := R2 - R2;  B3 := R2;
  END;
  COMMENT STORE MODIFIED IAW;
  RESET(CHARADDR);  R4 := OPERAND;  OPERAND := IAWADDR;  R2 := R6;
  R1 := 2;  MEMORY;  OPERAND := R4;  SET(CHARADDR);
END;
FINALY:R7 := INST AND #3800000 SHRL 21;  R6 := INST SHLL 9 SHRL 29;
```

```
R5 := CPUBASE + ACCG + R7;
IF OPCODE = 0 THEN BEGIN R1 := 2;  R2 := R1; INTERRUPT;   END;
IF R7 = 28 THEN  R4 := _28 ELSE R4 := 4;  ANEXT := R4;
R1 := 1;
COMMENT R9 = INSTRUCTION WORD, CPUBASE = CPU GROUP BASE, R8 = OP CODE,
  R10 = OPERAND/ADDRESS, R12 = ACTIVE STATUS REGISTER, R7 = ACCUMULATOR
  A DESIGNATOR, R6 = K DESIGNATOR, R5 = ACCUM A ADDRESS, R1 = 1;
IF TRACE THEN WTRACE;
IF OPCODE >= 40 THEN
BEGIN
  FULLWORD3;
  IF REPEAT THEN GOTO EXEC ELSE GOTO SETUP;
END;
IF OPCODE >= 7 THEN FULLWORD2 ELSE FULLWORD;
IF REPEAT THEN GOTJ EXEC ELSE GOTO SETUP;
BOMB: DUMP;
MVC(131,WBUF,BLANK):  R0 := @WBUF;  WRITE;
MVC(22,WBUF,"**** END EXECUTION ****");  WRITE;
WIPEOUT: IF PAGING THEN
  BEGIN
    R0 := _1;  GETPAGE;  COMMENT CLOSE OUT PAGE FILE;
  END;
END.
```

BIBLIOGRAPHY

1. Univac Defense Systems Division, Sperry Rand Corporation, Computer Set AN/UYK-7 (V) Technical Manual, v. 1, January 1971, March 1972 (Change 1).

2. Univac Division, Sperry Rand Corporation, AN/UYK-7 Military Computer Technical Description, no date.

3. Malcom. M. A., PL 360 (Revised), A Programming Language for the IBM 360, Stanford Univ., May 1971.

4. International Business Machines Corporation, IBM System/360 Principles of Operation, 8th ed., September 1968.

INITIAL DISTRIBUTION LIST

No. Copies

1. Defense Documentation Center 2
Cameron Station
Alexandria, Virginia 22314

2. Library, Code 0212 2
Naval Postgraduate School
Monterey, California 93940

3. Instructor Raymond H. Brubaker, Jr., Code 53Bh 1
Department of Mathematics
Naval Postgraduate School
Monterey, California 93940

4. Asst Professor Gordon H. Syms, Code 53Zz 1
Department of Mathematics
Naval Postgraduate School
Monterey, California 93940

5. Professor Gerald L. Barksdale, Jr., Code 72Bv 1
Computer Science Group
Naval Postgraduate School
Monterey, California 93940

6. LCDR Frederick C. Powell, USN 1
FOCCPAC
FPO San Francisco, California 96617

7. Major Allen B. Webb, USMC 1
588 A Sampson Lane
Monterey, California 93940

8. LT James Richardson King, USNR 1
1309 Octavia Street
New Orleans, Lousiana 70115

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Naval Postgraduate School Monterey, California 93940	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

AN AN/UYK-7 Interpreter Implemented On The IBM System/360

4. DESCRIPTIVE NOTES (Type of report and, inclusive dates)
Master's Thesis; December 1972

5. AUTHOR(S) (First name, middle initial, last name)
Frederick C. Powell
Allen B. Webb
James R. King

6. REPORT DATE December 1972	7a. TOTAL NO. OF PAGES 136	7b. NO. OF REFS
8a. CONTRACT OR GRANT NO. b. PROJECT NO. c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

An IBM System/360 based interpreter program for the AN/UYK-7 multiprocessing computer system is presented. The program provides interpretive execution of the AN/UYK-7 instruction set, an interrupt-handling capability, a variable-sized simulated memory and a multiprocessing capability for up to three central processors. Unput/output is limited to a card reader and a line printer. Various segments of the program are discussed and a detailed User's Manual is provided.

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
AN/UYK-7						
Interpreter						
Multiprocessing						
Paging						

Thesis
P762
c.1

Powell

An AN/UYK-7 interpreter implemented on the IBM System/360.

Thesis
P762
c.1

141404

Powell

An AN/UYK-7 interpreter implemented on the IBM System/360.